



YAYASAN PRIMA AGUS TEKNIK



Manajemen Pengetahuan Kecerdasan Buatan (AI)



Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM



Manajemen Pengetahuan Kecerdasan Buatan (AI)



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK

Jl. Majapahit No. 605 Semarang

Telp. (024) 6723456. Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

ISBN 978-634-7227-29-4 (PDF)



9

786347

227294

Manajemen Pengetahuan Kecerdasan Buatan (AI)

Penulis :

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

ISBN : 978-634-7227-29-4

Editor :

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

Penyunting :

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :

Irdha Yuniarto, S.Ds., M.Kom

Penebit :

Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi :

Jl. Majapahit no 605 Semarang

Telp. 08122925000

Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :

Universitas STEKOM

Jl. Majapahit no 605 Semarang

Telp. 08122925000

Fax. 024-6710144

Email : info@stekom.ac.id

Hak cipta dilindungi undang-undang

Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara
apapun tanpa ijin dari penulis

KATA PENGANTAR

Puji dan syukur Penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, sehingga buku berjudul *Manajemen Pengetahuan Kecerdasan Buatan (AI)* ini dapat terselesaikan dengan baik. Buku ini disusun sebagai bahan ajar dan referensi bagi mahasiswa, peneliti, dan praktisi yang ingin memahami konsep serta implementasi manajemen pengetahuan dalam bidang kecerdasan buatan secara mendalam dan sistematis. Dengan semakin berkembangnya teknologi AI, pengelolaan dan representasi pengetahuan menjadi sangat krusial untuk meningkatkan kecerdasan dan efisiensi sistem yang dibangun. Oleh karena itu, Kami berharap buku ini dapat menjadi referensi utama bagi para mahasiswa, peneliti, dan praktisi yang ingin mengembangkan kompetensi mereka dalam bidang manajemen pengetahuan kecerdasan buatan. Dengan pemahaman yang diperoleh melalui buku ini, diharapkan pembaca mampu merancang dan mengimplementasikan sistem AI yang lebih cerdas, adaptif, dan efektif.

Buku ini terdiri atas sembilan bab yang membahas mulai dari perencanaan klasik, algoritma pencarian ruang-keadaan, representasi pengetahuan, penalaran probabilistik, hingga pembelajaran mesin dan pengambilan keputusan kompleks. Setiap bab dirancang untuk memberikan gambaran teoritis yang kuat sekaligus aplikatif, sehingga pembaca dapat memperoleh pemahaman yang komprehensif dan dapat diaplikasikan dalam penelitian maupun pengembangan teknologi AI.

Penulis menyampaikan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan bantuan, dukungan, serta masukan berharga selama proses penyusunan buku ini, baik secara langsung maupun tidak langsung. Kritik dan saran yang membangun sangat Penulis harapkan demi penyempurnaan buku ini di masa mendatang.

Semoga buku ini dapat memberikan manfaat dan kontribusi positif dalam pengembangan ilmu pengetahuan, khususnya dalam pengelolaan pengetahuan dan kecerdasan buatan.

Semarang, Agustus 2025

Penulis

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM.

DAFTAR ISI

Halaman Judul	i
Kata Pengantar	ii
Daftar Isi	iii
BAB 1 PERENCANAAN KLASIK	1
1.1 Definisi Perencanaan Klasik	1
1.2 Algoritma Untuk Perencanaan Sebagai Pencarian Ruang-Kadaan.....	7
1.3 Grafik Perencanaan	15
1.4 Pendekatan Perencanaan Klasik Lainnya.....	24
1.5 Analisis Pendekatan Perencanaan	30
1.6 Ringkasan.....	31
BAB 2 PERENCANAAN DAN TINDAKAN DI DUNIA NYATA	32
2.1 Waktu, Jadwal, Dan Sumber Daya	32
2.2 Perencanaan Hirarkis.....	37
2.3 Perencanaan Dan Tindakan Di Wilayah Nondeterministik.....	47
2.4 Perencanaan Multiagen.....	59
2.5 Ringkasan.....	65
BAB 3 REPRESENTASI PENGETAHUAN.....	67
3.1 Rekayasa Ontologi	67
3.2 Kategori Dan Objek.....	70
3.3 Event Calculus.....	77
3.4 Kejadian Mental Dan Objek Mental	82
3.5 Sistem Penalaran Untuk Kategori	86
3.6 Penalaran Dengan Informasi Default.....	91
3.7 Dunia Belanja Internet.....	96
3.8 Ringkasan.....	101
BAB 4 MENGUKUR KETIDAKPASTIAN	103
4.1 Bertindak Di Bawah Ketidakpastian	103
4.2 Notasi Probabilitas Dasar	107
4.3 Inferensi Menggunakan Distribusi Joint Lengkap.....	115
4.4 Kemandirian	119
4.5 Aturan Bayes Dan Penggunaannya.....	121
4.6 Dunia Wumpus Dikunjungi Kembali	126
4.7 Ringkasan.....	129
BAB 5 PENALARAN PROBABILISTIK	131
5.1 Mewakili Pengetahuan Dalam Domain Yang Tak Pasti	131
5.2 Semantik Jaringan Bayesian	134
5.3 Representasi Distribusi Bersyarat Yang Efisien	140
5.4 Inferensi Tepat Dalam Jaringan Bayesian	144
5.5 Inferensi Perkiraan Dalam Jaringan Bayesian.....	153
5.6 Model Kemungkinan Relasional Dan Orde Pertama	164
5.7 Pendekatan Lain Untuk Penalaran Yang Tak Pasti	171
5.8 Ringkasan.....	178
BAB 6 PENALARAN PROBABILISTIK DARI WAKTU KE WAKTU	179
6.1 Waktu Dan Ketidakpastian	179

6.2	Inferensi Dalam Model Temporal	183
6.3	Model Markov Tersembunyi.....	193
6.4	Filter Kalman.....	199
6.5	Jaringan Bayesian Dinamis	207
6.6	Melacak Banyak Objek	217
6.7	Ringkasan.....	221
BAB 7	MEMBUAT KEPUTUSAN SEDERHANA	223
7.1	Menggabungkan Keyakinan Dan Keinginan Dalam Ketidakpastian	223
7.2	Dasar Teori Utilitas	224
7.3	Fungsi Utilitas	227
7.4	Fungsi Utilitas Multiatribut	235
7.5	Jaringan Keputusan	240
7.6	Nilai Informasi	243
7.7	Sistem Pakar Teori Keputusan	247
7.8	Ringkasan.....	251
BAB 8	MEMBUAT KEPUTUSAN KOMPLEKS	252
8.1	Masalah Keputusan Sekuensial	252
8.2	Iterasi Nilai.....	260
8.3	Iterasi Kebijakan	264
8.4	MDPS Yang Dapat Diamat Sebagian	267
8.5	Keputusan Dengan Banyak Agen: Teori Permainan	276
8.6	Desain Mekanisme	291
8.7	Ringkasan.....	298
BAB 9	BELAJAR DARI CONTOH	300
9.1	Bentuk Pembelajaran	300
9.2	Pembelajaran Terawasi.....	302
9.3	Pohon Keputusan Pembelajaran	305
9.4	Mengevaluasi Dan Memilih Hipotesis Terbaik	316
9.5	Teori Pembelajaran.....	323
9.6	Regresi Dan Klasifikasi Dengan Model Linier.....	328
9.7	Jaringan Syaraf Tiruan	339
9.8	Model Nonparametrik.....	350
9.9	Mesin Vektor Pendukung	358
9.10	Pembelajaran Ensemble	362
9.11	Pembelajaran Mesin Praktis	368
9.12	Kesimpulan	372
DAFTAR PUSTAKA		374

BAB 1

PERENCANAAN KLASIK

1.1 DEFINISI PERENCANAAN KLASIK

Agen pemecahan dapat menemukan urutan tindakan yang menghasilkan status tujuan. Namun, ia berurusan dengan representasi atomik dari status dan dengan demikian membutuhkan heuristik khusus domain yang baik agar dapat bekerja dengan baik. Agen logika proposisional hibrida dapat menemukan rencana tanpa heuristik khusus domain karena ia menggunakan heuristik domain-independen berdasarkan struktur logis masalah. Namun, hal itu bergantung pada inferensi proposisional dasar (bebas variabel), yang berarti bahwa hal itu dapat dibanjiri ketika terdapat banyak tindakan dan keadaan. Misalnya, dalam dunia wumpus, tindakan sederhana untuk melangkah maju harus diulang untuk keempat orientasi agen, langkah waktu T , dan n^2 lokasi saat ini.

Sebagai tanggapan terhadap hal ini, para peneliti perencanaan telah menetapkan representasi yang difaktorkan—yang di dalamnya keadaan dunia diwakili oleh sekumpulan variabel. Kami menggunakan bahasa yang disebut PDDL, Planning Domain Definition Language, yang memungkinkan kami untuk mengekspresikan semua $4Tn^2$ tindakan dengan satu skema tindakan. Ada beberapa versi PDDL; kami memilih versi yang sederhana dan mengubah sintaksisnya agar konsisten dengan bagian buku lainnya. Sekarang kami menunjukkan bagaimana PDDL menjelaskan empat hal yang kami perlukan untuk mendefinisikan masalah pencarian: keadaan awal, tindakan yang tersedia dalam suatu keadaan, hasil penerapan tindakan, dan uji sasaran.

Setiap keadaan direpresentasikan sebagai konjungsi fluents yang merupakan atom dasar yang tidak berfungsi. Misalnya, $Poor \wedge Unknown$ mungkin mewakili keadaan agen yang malang, dan keadaan dalam masalah pengiriman paket mungkin adalah $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$. Semantik basis data digunakan: asumsi dunia tertutup berarti bahwa semua fluent yang tidak disebutkan adalah salah, dan asumsi nama unik berarti bahwa $Truk_1$ dan $Truk_2$ berbeda. Fluent berikut tidak diperbolehkan dalam suatu status: $At(x, y)$ (karena bukan dasar), $\neg Poor$ (karena merupakan negasi), dan $At(Father(Fred), Sydney)$ (karena menggunakan simbol fungsi). Representasi status dirancang dengan cermat sehingga suatu status dapat diperlakukan baik sebagai konjungsi fluent, yang dapat dimanipulasi dengan inferensi logis, atau sebagai sekumpulan fluent, yang dapat dimanipulasi dengan operasi himpunan. Semantik himpunan terkadang lebih mudah ditangani.

Tindakan dijelaskan oleh sekumpulan skema tindakan yang secara implisit mendefinisikan fungsi $ACTION_{(s)}$ dan $RESULT_{(s,a)}$ yang diperlukan untuk melakukan pencarian pemecahan masalah. Kita melihat bahwa sistem apa pun untuk deskripsi tindakan perlu memecahkan masalah kerangka—untuk mengatakan apa yang berubah dan apa yang tetap sama sebagai hasil tindakan.

Perencanaan klasik berfokus pada masalah di mana sebagian besar tindakan tidak mengubah sebagian besar hal. Bayangkan dunia yang terdiri dari sekumpulan objek pada permukaan datar. Tindakan mendorong objek menyebabkan objek tersebut mengubah lokasinya dengan vektor Δ . Deskripsi singkat tentang tindakan tersebut harus menyebutkan hanya Δ ; tidak harus menyebutkan semua objek yang tetap di tempatnya. PDDL melakukannya dengan menentukan hasil tindakan dalam hal apa yang berubah; semua yang tetap sama tidak disebutkan.

Seperangkat tindakan dasar (bebas variabel) dapat direpresentasikan oleh satu skema tindakan. Skema tersebut adalah representasi yang ditingkatkan—ia meningkatkan tingkat penalaran dari logika proposisional ke subset terbatas dari logika orde pertama. Misalnya, berikut adalah skema tindakan untuk menerbangkan pesawat dari satu lokasi ke lokasi lain:

$$\begin{aligned} & \text{Action}(\text{Fly}(p, \text{from}, \text{to}), \\ & \text{PRECOND: } \text{At}(p, \text{from}) \wedge \text{Plane}(p) \wedge \text{Airport}(\text{from}) \wedge \text{Airport}(\text{to}) \\ & \text{EFFECT: } \neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to}) \end{aligned}$$

Skema tersebut terdiri dari nama tindakan, daftar semua variabel yang digunakan dalam skema, prasyarat, dan efek. Meskipun kami belum menjelaskan bagaimana skema tindakan diubah menjadi kalimat logis, anggaplah variabel-variabel tersebut dikuantifikasi secara universal. Kita bebas memilih nilai apa pun yang kita inginkan untuk mewujudkan variabel-variabel tersebut. Misalnya, berikut adalah satu tindakan dasar yang dihasilkan dari penggantian nilai untuk semua variabel:

$$\begin{aligned} & \text{Action}(\text{Fly}(P_1, \text{SFO}, \text{JFK}), \\ & \text{PRECOND: } \text{At}(P_1, \text{SFO}) \wedge \text{Plane}(P_1) \wedge \text{Airport}(\text{SFO}) \wedge \text{Airport}(\text{JFK}) \\ & \text{EFFECT: } \neg \text{At}(P_1, \text{SFO}) \wedge \text{At}(P_1, \text{JFK}) \end{aligned}$$

Prasyarat dan akibat dari suatu tindakan masing-masing merupakan konjungsi dari literal (kalimat atom positif atau negatif). Prasyarat mendefinisikan keadaan di mana tindakan dapat dieksekusi, dan akibat mendefinisikan hasil dari eksekusi tindakan. Suatu tindakan a dapat dieksekusi dalam keadaan s jika s memerlukan prasyarat dari a . Konsekuensinya juga dapat diungkapkan dengan semantik himpunan: $s \models q$ jika setiap literal positif dalam q ada dalam s dan setiap literal negatif dalam q tidak ada. Dalam notasi formal kita katakan

$$(a \in \text{ACTION}(s)) \Leftrightarrow s \models \text{PRECOND}(a),$$

di mana semua variabel dalam suatu bilangan dikuantifikasi secara universal. Misalnya,

$$\begin{aligned} & \forall p, \text{from}, \text{to} (\text{Fly}(p, \text{from}, \text{to}) \in \text{ACTION}(s)) \Leftrightarrow \\ & s \models (\text{At}(p, \text{from}) \wedge \text{Plane}(p) \wedge \text{Airport}(\text{from}) \wedge \text{Airport}(\text{to})) \end{aligned}$$

Kita katakan bahwa tindakan a dapat diterapkan dalam status s jika prasyaratnya dipenuhi oleh s . Ketika skema tindakan a berisi variabel, skema tersebut mungkin memiliki beberapa instansiasi yang dapat diterapkan. Misalnya, dengan status awal yang didefinisikan dalam Gambar 1.1, tindakan Fly dapat diwujudkan sebagai $\text{Fly}(P_1, \text{SFO}, \text{JFK})$ atau sebagai

$Fly(P_2, JFK, SFO)$, yang keduanya dapat diterapkan dalam status awal. Jika suatu tindakan a memiliki v variabel, maka, dalam domain dengan k nama objek yang unik, dibutuhkan waktu $O(v^k)$ dalam kasus terburuk untuk menemukan tindakan dasar yang dapat diterapkan.

Terkadang kita ingin mengajukan masalah PDDL—mengganti setiap skema tindakan dengan serangkaian tindakan dasar dan kemudian menggunakan penyelesai proposisional seperti SATPLAN untuk menemukan solusi. Namun, hal ini tidak praktis ketika v dan k besar. Hasil dari pelaksanaan aksi a dalam status s didefinisikan sebagai status s' yang direpresentasikan oleh himpunan fluent yang dibentuk dengan memulai dari s , menghapus fluent yang muncul sebagai literal negatif dalam efek aksi (yang kita sebut daftar hapus atau $DEL(a)$), dan menambahkan fluent yang merupakan literal positif dalam efek aksi (yang kita sebut daftar tambah atau $ADD(a)$):

$$RESULT(s, a) = (s - DEL(a)) \cup ADD(a). \quad (10.1)$$

Misalnya, dengan tindakan $Fly(P_1, SFO, JFK)$, kita akan menghapus $At(P_1, SFO)$ dan menambahkan $At(P_1, JFK)$. Merupakan persyaratan skema tindakan bahwa setiap variabel dalam efek juga harus muncul dalam kondisi awal. Dengan demikian, ketika kondisi awal dicocokkan dengan keadaan s , semua variabel akan terikat, dan $RESULT(s, a)$ akan memiliki atom dasar saja. Dengan kata lain, keadaan dasar tertutup di bawah operasi $RESULT$.

Perhatikan juga bahwa fluent tidak secara eksplisit merujuk ke waktu. Di sana kita memerlukan superskrip untuk waktu, dan aksioma keadaan penerus dalam bentuk:

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t).$$

Dalam PDDL, waktu dan status tersirat dalam skema tindakan: prasyarat selalu merujuk ke waktu t dan efek ke waktu $t + 1$.

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1)
    ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK)
    ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
Action(Load(c, p, a),
    PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
    EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
    PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
    EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
    PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
    EFFECT: ¬ At(p, from) ∧ At(p, to))
```

Gambar 1.1 Deskripsi PDDL tentang masalah perencanaan transportasi kargo udara.

Seperangkat skema tindakan berfungsi sebagai definisi domain perencanaan. Masalah khusus dalam domain didefinisikan dengan penambahan status awal dan tujuan. Status awal adalah konjungsi atom dasar. (Seperti halnya semua status, asumsi dunia tertutup digunakan, yang berarti bahwa atom apa pun yang tidak disebutkan adalah salah.) Tujuannya seperti prasyarat: konjungsi literal (positif atau negatif) yang mungkin berisi variabel, seperti $At(p, SFO) \wedge Plane(p)$.

Setiap variabel diperlakukan sebagai terkuantifikasi secara eksistensial, jadi tujuan ini adalah memiliki bidang apa pun di SFO. Masalah terpecahkan ketika kita dapat menemukan urutan tindakan yang berakhir pada status s yang memerlukan tujuan. Misalnya, keadaan $Rich \wedge Famous \wedge Miserable$ mensyaratkan tujuan $Rich \wedge Famous$, dan keadaan $Plane(Plane_1) \wedge At(Plane_1, SFO)$ mensyaratkan tujuan $At(p, SFO) \wedge Plane(p)$. Sekarang kita telah mendefinisikan perencanaan sebagai masalah pencarian: kita memiliki keadaan awal, fungsi ACTION, fungsi RESULT, dan uji tujuan. Kita akan melihat beberapa contoh masalah sebelum menyelidiki algoritma pencarian yang efisien.

Contoh: Transportasi Kargo Udara

Gambar 1.1 menunjukkan masalah transportasi kargo udara yang melibatkan pemuatan dan pembongkaran kargo serta menerbangkannya dari satu tempat ke tempat lain. Masalah tersebut dapat didefinisikan dengan tiga tindakan: *Load*, *Unload* dan *Fly*. Tindakan tersebut memengaruhi dua predikat: $In(c, p)$ berarti kargo c berada di dalam bidang p , dan $At(x, a)$ berarti objek x (baik pesawat maupun kargo) berada di bandara a . Perhatikan bahwa beberapa kehati-hatian harus dilakukan untuk memastikan predikat At dipertahankan dengan benar. Ketika sebuah pesawat terbang dari satu bandara ke bandara lain, semua kargo di dalam pesawat ikut bersamanya.

Dalam logika orde pertama, akan mudah untuk mengukur semua objek yang berada di dalam pesawat. Namun, PDDL dasar tidak memiliki kuantifier universal, jadi kita memerlukan solusi yang berbeda. Pendekatan yang kita gunakan adalah dengan mengatakan bahwa sepotong kargo berhenti berada di mana pun saat berada di dalam pesawat; kargo hanya menjadi di bandara baru saat diturunkan. Jadi, At benar-benar berarti "tersedia untuk digunakan di lokasi tertentu." Rencana berikut ini merupakan solusi untuk masalah tersebut:

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK), \\ Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)].$$

Terakhir, ada masalah tindakan palsu seperti $Fly(P_1, JFK, JFK)$, yang seharusnya tidak dapat dilakukan, tetapi memiliki efek yang bertentangan (menurut definisi, efeknya akan mencakup $At(P_1, JFK) \wedge \neg At(P_1, JFK)$). Masalah seperti itu biasanya diabaikan, karena jarang menyebabkan rencana yang salah dibuat. Pendekatan yang benar adalah menambahkan prasyarat ketidaksetaraan yang menyatakan bahwa bandara dari dan ke harus berbeda.

Contoh: Masalah Ban Serep

Pertimbangkan masalah mengganti ban kempes. Tujuannya adalah agar ban serep yang bagus terpasang dengan benar pada as mobil, di mana keadaan awalnya adalah ban kempes

pada as dan ban serep yang bagus di bagasi. Untuk membuatnya tetap sederhana, versi masalah kita adalah yang abstrak, tanpa mur roda yang lengket atau komplikasi lainnya.

Hanya ada empat tindakan: mengeluarkan ban serep dari bagasi, mengeluarkan ban kempes dari as, memasang ban serep pada as, dan meninggalkan mobil tanpa pengawasan semalaman. Kami berasumsi bahwa mobil diparkir di lingkungan yang sangat buruk, sehingga efek meninggalkannya semalaman adalah bannya hilang. Solusi untuk masalah ini adalah $[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$.

```

Init(Tire(Flat)  $\wedge$  Tire(Spare)  $\wedge$  At(Flat, Axle)  $\wedge$  At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(obj, loc),
  PRECOND: At(obj, loc)
  EFFECT:  $\neg$ At(obj, loc)  $\wedge$  At(obj, Ground))
Action(PutOn(t, Axle),
  PRECOND: Tire(t)  $\wedge$  At(t, Ground)  $\wedge$   $\neg$  At(Flat, Axle)
  EFFECT:  $\neg$  At(t, Ground)  $\wedge$  At(t, Axle))
Action(LeaveOvernight,
  PRECOND:
    EFFECT:  $\neg$  At(Spare, Ground)  $\wedge$   $\neg$  At(Spare, Axle)  $\wedge$   $\neg$  At(Spare, Trunk)
            $\wedge$   $\neg$  At(Flat, Ground)  $\wedge$   $\neg$  At(Flat, Axle)  $\wedge$   $\neg$  At(Flat, Trunk))

```

Gambar 1.2 Masalah ban serep sederhana.

Contoh: Dunia Balok

Salah satu domain perencanaan yang paling terkenal dikenal sebagai dunia balok. Domain ini terdiri dari sekumpulan balok berbentuk kubus yang diletakkan di atas meja. Balok-balok tersebut dapat ditumpuk, tetapi hanya satu balok yang dapat langsung diletakkan di atas balok lainnya. Lengan robot dapat mengambil balok dan memindahkannya ke posisi lain, baik di atas meja maupun di atas balok lainnya.

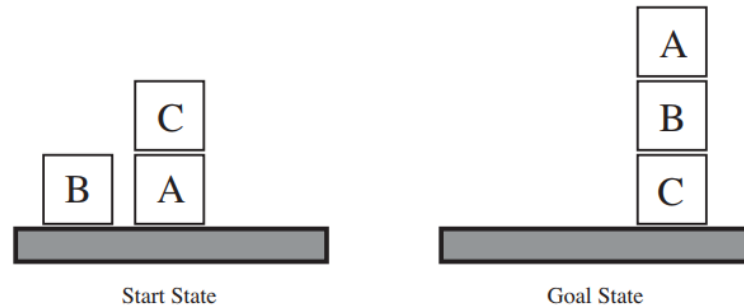
```

Init(On(A, Table)  $\wedge$  On(B, Table)  $\wedge$  On(C, A)  $\wedge$  Block(A)  $\wedge$  Block(B)  $\wedge$  Block(C)
       $\wedge$  Clear(B)  $\wedge$  Clear(C))
Goal(On(A, B)  $\wedge$  On(B, C))
Action(Move(b, x, y),
  PRECOND: On(b, x)  $\wedge$  Clear(b)  $\wedge$  Clear(y)  $\wedge$  Block(b)  $\wedge$  Block(y)  $\wedge$  (b  $\neq$  x)  $\wedge$ 
    (b  $\neq$  y)  $\wedge$  (x  $\neq$  y),
  EFFECT: On(b, y)  $\wedge$  Clear(x)  $\wedge$   $\neg$ On(b, x)  $\wedge$   $\neg$ Clear(y))
Action(MoveToTable(b, x),
  PRECOND: On(b, x)  $\wedge$  Clear(b)  $\wedge$  Block(b)  $\wedge$  (b  $\neq$  x),
  EFFECT: On(b, Table)  $\wedge$  Clear(x)  $\wedge$   $\neg$ On(b, x))

```

Gambar 1.3 Masalah perencanaan dalam dunia blok: membangun menara tiga blok. Salah satu solusinya adalah urutan $[MoveToTable(C, A), Move(B, Meja, C), Move(A, Meja, B)]$.

Lengan tersebut hanya dapat mengambil satu balok dalam satu waktu, jadi tidak dapat mengambil balok yang memiliki balok lain di atasnya. Sasarannya akan selalu membangun satu atau lebih tumpukan balok, yang ditentukan berdasarkan balok mana yang berada di atas balok lainnya. Misalnya, sasarannya mungkin adalah meletakkan balok *A* di atas *B* dan balok *B* di atas *C*.



Gambar 1.4 Diagram masalah blok-dunia pada Gambar 1.3.

Kita menggunakan $On(b, x)$ untuk menunjukkan bahwa balok b berada di atas x , di mana x adalah balok lain atau meja. Tindakan untuk memindahkan balok b dari atas x ke atas y adalah $Move(b, x, y)$. Sekarang, salah satu prasyarat untuk memindahkan b adalah tidak ada blok lain di atasnya. Dalam logika orde pertama, ini akan menjadi $\neg \exists x On(x, b)$ atau, sebagai alternatif, $\forall x \neg On(x, b)$. PDDL dasar tidak memperbolehkan kuantifier, jadi sebagai gantinya kami memperkenalkan predikat $Clear(x)$ yang benar ketika tidak ada apa pun di x .

Tindakan $Move$ memindahkan blok b dari x ke y jika b dan y kosong. Setelah gerakan dilakukan, b masih kosong tetapi y tidak. Upaya pertama pada skema $Move$ adalah

$$\begin{aligned} &Action(Move(b, x, y), \\ &\quad PRECOND: On(b, x) \wedge Clear(b) \wedge Clear(y), \\ &\quad EFFECT: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)). \end{aligned}$$

Sayangnya, ini tidak mempertahankan $Clear$ dengan benar saat x atau y adalah tabel. Saat x adalah $Table$, tindakan ini memiliki efek $Clear(Table)$, tetapi tabel tidak boleh menjadi clear; dan saat $y = Table$, ia memiliki prasyarat $Clear(Table)$, tetapi tabel tidak harus clear agar kita dapat memindahkan blok ke dalamnya. Untuk memperbaikinya, kita melakukan dua hal. Pertama, kita memperkenalkan tindakan lain untuk memindahkan blok b dari x ke tabel:

$$\begin{aligned} &Action(MoveToTable(b, x), \\ &\quad PRECOND: On(b, x) \wedge Clear(b), \\ &\quad EFFECT: On(b, Table) \wedge Clear(x) \wedge \neg On(b, x)). \end{aligned}$$

Kedua, kita ambil interpretasi $Clear(x)$ sebagai "ada ruang kosong di x untuk menampung blok." Berdasarkan interpretasi ini, $Clear(Table)$ akan selalu benar. Satu-satunya masalah adalah tidak ada yang mencegah perencanaan menggunakan $Move(b, x, Table)$ alih-alih

$MoveToTable(b, x)$. Kita bisa menerima masalah ini—ini akan menghasilkan ruang pencarian yang lebih besar dari yang diperlukan, tetapi tidak akan menghasilkan jawaban yang salah—atau kita bisa memperkenalkan predikat $Block$ dan menambahkan $Block(b) \wedge Block(y)$ ke prakondisi $Move$.

Kompleksitas Perencanaan Klasik

Dalam subbagian ini, kami mempertimbangkan kompleksitas teoretis perencanaan dan membedakan dua masalah keputusan. PlanSAT adalah pertanyaan tentang apakah ada rencana yang memecahkan masalah perencanaan. PlanSAT Terbatas menanyakan apakah ada solusi dengan panjang k atau kurang; ini dapat digunakan untuk menemukan rencana yang optimal.

Hasil pertama adalah bahwa kedua masalah keputusan dapat diputuskan untuk perencanaan klasik. Buktinya mengikuti fakta bahwa jumlah status terbatas. Namun, jika kita menambahkan simbol fungsi ke bahasa, maka jumlah status menjadi tak terbatas, dan PlanSAT hanya menjadi semi-dapat diputuskan: ada algoritme yang akan berakhir dengan jawaban yang benar untuk masalah yang dapat dipecahkan, tetapi mungkin tidak berakhir pada masalah yang tidak dapat dipecahkan. Masalah PlanSAT Terbatas tetap dapat diputuskan bahkan dengan adanya simbol fungsi.

Untuk bukti pernyataan dalam bagian ini. Baik PlanSAT maupun Bounded PlanSAT berada dalam kelas kompleksitas PSPACE, kelas yang lebih besar (dan karenanya lebih sulit) daripada NP dan merujuk pada masalah yang dapat dipecahkan oleh mesin Turing deterministik dengan jumlah ruang polinomial. Bahkan jika kita membuat beberapa pembatasan yang agak ketat, masalah tersebut tetap cukup sulit. Misalnya, jika kita tidak mengizinkan efek negatif, kedua masalah tersebut masih NP-keras. Namun, jika kita juga tidak mengizinkan prasyarat negatif, PlanSAT direduksi menjadi kelas P .

Hasil kasus terburuk ini mungkin tampak mengecewakan. Kita dapat merasa terhibur dengan fakta bahwa agen biasanya tidak diminta untuk menemukan rencana untuk contoh masalah kasus terburuk yang sewenang-wenang, tetapi diminta untuk membuat rencana dalam domain tertentu (seperti masalah blok-dunia dengan n blok), yang dapat jauh lebih mudah daripada kasus terburuk teoretis. Untuk banyak domain (termasuk dunia blok dan dunia kargo udara), Bounded PlanSAT adalah NP-lengkap sementara PlanSAT berada di P ; dengan kata lain, perencanaan optimal biasanya sulit, tetapi perencanaan sub-optimal terkadang mudah. Untuk berhasil dalam masalah yang lebih mudah daripada yang terburuk, kita memerlukan heuristik pencarian yang baik. Itulah keuntungan sebenarnya dari formalisme perencanaan klasik: formalisme ini telah memfasilitasi pengembangan heuristik domain-independen yang sangat akurat, sedangkan sistem yang didasarkan pada aksioma status penerus dalam logika orde pertama kurang berhasil dalam menghasilkan heuristik yang baik.

1.2 ALGORITMA UNTUK PERENCANAAN SEBAGAI PENCARIAN RUANG-KEADAAN

Sekarang kita alihkan perhatian kita ke algoritma perencanaan. Kita melihat bagaimana deskripsi masalah perencanaan mendefinisikan masalah pencarian: kita dapat mencari dari keadaan awal melalui ruang keadaan, mencari tujuan. Salah satu keuntungan bagus dari

representasi deklaratif skema tindakan adalah kita juga dapat mencari mundur dari tujuan, mencari keadaan awal. Gambar 1.5 membandingkan pencarian maju dan mundur.

Pencarian Ruang-Keadaan Maju (Progresi)

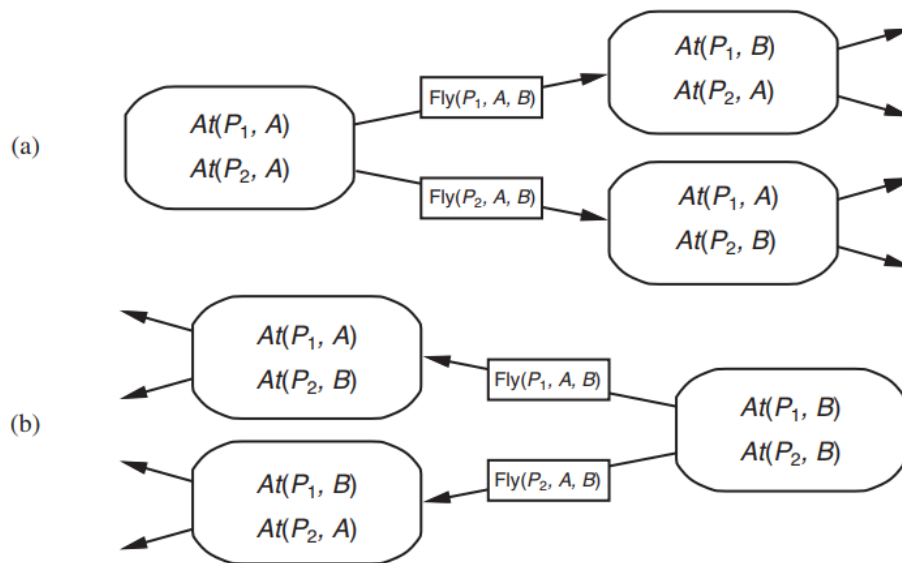
Sekarang setelah kita menunjukkan bagaimana masalah perencanaan dipetakan menjadi masalah pencarian, kita dapat memecahkan masalah perencanaan dengan salah satu algoritma pencarian heuristik atau algoritma pencarian lokal (asalkan kita melacak tindakan yang digunakan untuk mencapai tujuan). Dari hari-hari awal penelitian perencanaan (sekitar tahun 1961) hingga sekitar tahun 1998 diasumsikan bahwa pencarian ruang-keadaan maju terlalu tidak efisien untuk menjadi praktis. Tidak sulit untuk menemukan alasannya.

Pertama, penelusuran ke depan cenderung mengeksplorasi tindakan yang tidak relevan. Pertimbangkan tugas mulia membeli salinan *AI: A Modern Approach* dari penjual buku daring. Misalkan ada skema tindakan *Buy(isbn)* dengan efek *Own(isbn)*. ISBN terdiri dari 10 digit, jadi skema tindakan ini mewakili 10 miliar tindakan di darat. Algoritme penelusuran ke depan yang tidak memiliki informasi harus mulai menghitung 10 miliar tindakan ini untuk menemukan satu yang mengarah ke tujuan.

Kedua, masalah perencanaan sering kali memiliki ruang keadaan yang besar. Pertimbangkan masalah kargo udara dengan 10 bandara, di mana setiap bandara memiliki 5 pesawat dan 20 buah kargo. Tujuannya adalah untuk memindahkan semua kargo di bandara A ke bandara B. Ada solusi sederhana untuk masalah tersebut: memuat 20 buah kargo ke salah satu pesawat di A, menerbangkan pesawat ke B, dan membongkar kargo.

Menemukan solusinya bisa jadi sulit karena faktor percabangan rata-rata sangat besar: masing-masing dari 50 pesawat dapat terbang ke 9 bandara lain, dan masing-masing dari 200 paket dapat diturunkan (jika sudah dimuat) atau dimuat ke pesawat mana pun di bandaranya (jika sudah dibongkar). Jadi di negara bagian mana pun, ada minimal 450 tindakan (ketika semua paket berada di bandara tanpa pesawat) dan maksimal 10.450 (ketika semua paket dan pesawat berada di bandara yang sama). Rata-rata, katakanlah ada sekitar 2000 tindakan yang mungkin per negara bagian, jadi grafik pencarian hingga kedalaman solusi yang jelas memiliki sekitar 2000⁴¹ simpul.

Jelas, bahkan contoh masalah yang relatif kecil ini tidak ada harapan tanpa heuristik yang akurat. Meskipun banyak aplikasi perencanaan di dunia nyata telah bergantung pada heuristik khusus domain, bahwa heuristik independen domain yang kuat dapat diturunkan secara otomatis; itulah yang membuat pencarian ke depan layak dilakukan.



Gambar 1.5 Dua pendekatan untuk mencari rencana. (a) Pencarian maju (progresi) melalui ruang keadaan, dimulai dari keadaan awal dan menggunakan tindakan masalah untuk mencari anggota himpunan keadaan tujuan ke depan. (b) Pencarian mundur (regresi) melalui himpunan keadaan yang relevan, dimulai dari himpunan keadaan yang mewakili tujuan dan menggunakan kebalikan dari tindakan untuk mencari mundur keadaan awal.

Pencarian Status-Relevan Mundur (Regresi)

Dalam pencarian regresi, kita mulai dari tujuan dan menerapkan tindakan mundur hingga kita menemukan serangkaian langkah yang mencapai status awal. Ini disebut pencarian status-relevan karena kita hanya mempertimbangkan tindakan yang relevan dengan tujuan (atau status saat ini). Seperti dalam pencarian status-keyakinan, ada serangkaian status relevan yang perlu dipertimbangkan di setiap langkah, bukan hanya satu status.

Kita mulai dengan tujuan, yang merupakan konjungsi literal yang membentuk deskripsi serangkaian status—misalnya, tujuan $\neg Poor \wedge Famous$ menggambarkan status-status di mana *Poor* salah, *Famous* benar, dan fluent lainnya dapat memiliki nilai apa pun. Jika ada n fluent dasar dalam domain, maka ada 2^n status dasar (setiap fluent dapat benar atau salah), tetapi 3^n deskripsi serangkaian status tujuan (setiap fluent dapat positif, negatif, atau tidak disebutkan). Secara umum, pencarian mundur hanya berfungsi jika kita tahu cara melakukan regresi dari deskripsi status ke deskripsi status pendahulu.

Misalnya, sulit untuk mencari mundur untuk solusi masalah n -ratu karena tidak ada cara mudah untuk mendeskripsikan status yang berjarak satu langkah dari tujuan. Untungnya, representasi PDDL dirancang untuk memudahkan regresi tindakan—jika domain dapat diekspresikan dalam PDDL, maka kita dapat melakukan pencarian regresi padanya. Diberikan deskripsi tujuan dasar g dan tindakan dasar a , regresi dari g atas a memberi kita deskripsi status g' yang didefinisikan oleh

$$g' = (g - ADD(a)) \cup Precond(a).$$

Artinya, efek yang ditambahkan oleh tindakan tidak perlu benar sebelumnya, dan juga prasyarat harus berlaku sebelumnya, atau tindakan tidak dapat dieksekusi. Perhatikan bahwa $DEL(a)$ tidak muncul dalam rumus; itu karena walaupun kita tahu bahwa kata-kata fasih dalam $DEL(a)$ tidak lagi benar setelah tindakan, kita tidak tahu apakah kata-kata itu benar sebelumnya atau tidak, jadi tidak ada yang bisa dikatakan tentang kata-kata itu.

Untuk mendapatkan keuntungan penuh dari pencarian mundur, kita perlu menangani tindakan dan status yang sebagian tidak terwujud, bukan hanya yang mendasar. Misalnya, anggaplah tujuannya adalah untuk mengirimkan kargo tertentu ke $SFO: At(C_2, SFO)$. Itu menunjukkan tindakan $Unload(C_2, p', SFO)$:

Action($Unload(C_2, p', SFO)$,
 PRECOND: $In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$
 EFFECT: $At(C_2, SFO) \wedge \neg In(C_2, p')$).

(Perhatikan bahwa kita memiliki nama variabel yang distandardisasi (mengubah p menjadi p' dalam kasus ini) sehingga tidak akan ada kebingungan antara nama variabel jika kita kebetulan menggunakan skema tindakan yang sama dua kali dalam satu rencana. Ini merupakan pembongkaran paket dari bidang yang tidak ditentukan di SFO; bidang apa pun bisa digunakan, tetapi kita tidak perlu mengatakan yang mana sekarang. Kita dapat memanfaatkan kekuatan representasi orde pertama: satu deskripsi merangkum kemungkinan penggunaan bidang mana pun dengan mengkuantifikasi secara implisit atas p' . Deskripsi keadaan yang diregresi adalah

$$g' = In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO).$$

Masalah terakhir adalah memutuskan tindakan mana yang akan diulang. Dalam arah maju, kami memilih tindakan yang dapat diterapkan—tindakan yang dapat menjadi langkah berikutnya dalam rencana. Dalam pencarian mundur, kami menginginkan tindakan yang relevan—tindakan yang dapat menjadi langkah terakhir dalam rencana yang mengarah ke status tujuan saat ini.

Agar suatu tindakan relevan dengan suatu tujuan, tindakan tersebut jelas harus berkontribusi pada tujuan tersebut: setidaknya satu dari efek tindakan tersebut (baik positif maupun negatif) harus menyatu dengan elemen tujuan. Yang kurang jelas adalah bahwa tindakan tersebut tidak boleh memiliki efek (positif maupun negatif) yang meniadakan elemen tujuan.

Sekarang, jika tujuannya adalah $A \wedge B \wedge C$ dan suatu tindakan memiliki efek $A \wedge B \wedge \neg C$, maka ada pengertian sehari-hari bahwa tindakan tersebut sangat relevan dengan tujuan—tindakan tersebut membawa kita dua pertiga dari tujuan tersebut. Namun, hal ini tidak relevan dalam pengertian teknis yang didefinisikan di sini, karena tindakan ini tidak dapat

menjadi langkah terakhir dari sebuah solusi—kita akan selalu membutuhkan setidaknya satu langkah lagi untuk mencapai C .

Dengan tujuan $At(C_2, SFO)$, beberapa perwujudan *Unload* relevan: kita dapat memilih bidang tertentu untuk melakukan unload, atau kita dapat membiarkan bidang tersebut tidak ditentukan dengan menggunakan tindakan $Unload(C_2, p', SFO)$. Kita dapat mengurangi faktor percabangan tanpa mengesampingkan solusi apa pun dengan selalu menggunakan tindakan yang dibentuk dengan mensubstitusikan pemersatu yang paling umum ke dalam skema tindakan (yang terstandarisasi).

Sebagai contoh lain, pertimbangkan tujuan $Own(0136042597)$, dengan status awal dengan 10 miliar *ISBNs*, dan skema tindakan tunggal

$$A = Action(Buy(i), PRECOND: ISBN(i), EFFECT: Own(i)).$$

Seperti yang telah kami sebutkan sebelumnya, penelusuran maju tanpa heuristik harus mulai menghitung 10 miliar tindakan *Buy* dasar. Namun dengan penelusuran mundur, kami akan menyatukan tujuan $Own(0136042597)$ dengan efek (terstandarisasi) $Own(i')$, menghasilkan substitusi $\theta = \{i'/0136042597\}$. Kemudian kami akan melakukan regresi atas tindakan $Subst(\theta, A')$ untuk menghasilkan deskripsi status pendahulu $ISBN(0136042597)$. Ini adalah bagian dari, dan dengan demikian disyaratkan oleh, status awal, jadi kami sudah selesai.

Kita dapat membuatnya lebih formal. Asumsikan deskripsi tujuan g yang berisi literal tujuan gi dan skema tindakan A yang distandarkan untuk menghasilkan A' . Jika A' memiliki literal efek e'_j di mana $Unify(gi, e'_j) = \theta$ dan di mana kita mendefinisikan $a' = SUBST(\theta, A')$ dan jika tidak ada efek a' dalam a yang merupakan negasi literal a dalam g , maka a adalah tindakan yang relevan terhadap g .

Pencarian mundur menjaga faktor percabangan lebih rendah daripada pencarian maju, untuk sebagian besar domain masalah. Namun, fakta bahwa pencarian mundur menggunakan set status daripada status individual membuatnya lebih sulit untuk menghasilkan heuristik yang baik. Itulah alasan utama mengapa sebagian besar sistem saat ini lebih menyukai pencarian maju.

Heuristik untuk Perencanaan

Pencarian maju maupun mundur tidak efisien tanpa fungsi heuristik yang baik. Fungsi heuristik $h(s)$ memperkirakan jarak dari status s ke tujuan dan jika kita dapat memperoleh heuristik yang dapat diterima untuk jarak ini—yang tidak melebihi-lebihkan—maka kita dapat menggunakan pencarian A* untuk menemukan solusi optimal. Heuristik yang dapat diterima dapat diperoleh dengan mendefinisikan masalah yang lebih mudah dipecahkan. Biaya pasti dari solusi untuk masalah yang lebih mudah ini kemudian menjadi heuristik untuk masalah awal. Menurut definisi, tidak ada cara untuk menganalisis status atomik, dan dengan demikian diperlukan kecerdasan oleh analis manusia untuk mendefinisikan heuristik khusus domain yang baik untuk masalah pencarian dengan status atomik. Perencanaan menggunakan representasi terfaktor untuk status dan skema tindakan.

Hal itu memungkinkan untuk mendefinisikan heuristik independen domain yang baik dan bagi program untuk secara otomatis menerapkan heuristik independen domain yang baik untuk masalah tertentu. Bayangkan masalah pencarian sebagai grafik yang simpulnya adalah status dan sisinya adalah tindakan. Masalahnya adalah menemukan jalur yang menghubungkan status awal ke status tujuan. Ada dua cara untuk melonggarkan masalah ini agar lebih mudah: dengan menambahkan lebih banyak sisi ke grafik, sehingga lebih mudah menemukan jalur, atau dengan mengelompokkan beberapa simpul bersama-sama, membentuk abstraksi ruang status yang memiliki lebih sedikit status, dan dengan demikian lebih mudah dicari.

Pertama-tama kita lihat heuristik yang menambahkan sisi ke grafik. Misalnya, heuristik abaikan kondisi awal membuang semua kondisi awal dari tindakan. Setiap tindakan menjadi berlaku di setiap status, dan setiap tujuan tunggal yang lancar dapat dicapai dalam satu langkah (jika ada tindakan yang berlaku—jika tidak, masalahnya tidak mungkin). Ini hampir menyiratkan bahwa jumlah langkah yang diperlukan untuk memecahkan masalah yang dilonggarkan adalah jumlah tujuan yang tidak terpenuhi—hampir tetapi tidak sepenuhnya, karena (1) beberapa tindakan dapat mencapai beberapa tujuan dan (2) beberapa tindakan dapat membatalkan efek dari yang lain. Untuk banyak masalah, heuristik yang akurat diperoleh dengan mempertimbangkan (1) dan mengabaikan (2). Pertama, kita melonggarkan tindakan dengan menghapus semua prasyarat dan semua efek kecuali yang merupakan literal dalam tujuan. Kemudian, kita hitung jumlah minimum tindakan yang diperlukan sedemikian rupa sehingga gabungan efek tindakan tersebut memenuhi tujuan. Ini adalah contoh dari masalah set-cover. Ada satu gangguan kecil: masalah set-cover bersifat NP-keras. Untungnya, algoritma greedy sederhana dijamin akan mengembalikan set covering yang ukurannya berada dalam faktor $\log n$ dari minimum covering yang sebenarnya, di mana n adalah jumlah literal dalam tujuan. Sayangnya, algoritma greedy kehilangan jaminan penerimaan.

Dimungkinkan juga untuk mengabaikan hanya prasyarat tindakan yang dipilih. Pertimbangkan teka-teki blok geser (teka-teki 8 atau teka-teki 15). Kita dapat mengodekan ini sebagai masalah perencanaan yang melibatkan petak dengan satu skema *Slide*:

$$\begin{aligned} & \text{Action}(\text{Slide}(t, s_1, s_2), \\ & \quad \text{PRECOND: } On(t, s_1) \wedge \text{Tile}(t) \wedge \text{Blank}(s_2) \wedge \text{Adjacent}(s_1, s_2) \\ & \quad \text{EFFECT: } On(t, s_2) \wedge \text{Blank}(s_1) \wedge \neg On(t, s_1) \wedge \neg \text{Blank}(s_2)) \end{aligned}$$

Jika kita menghapus prasyarat $\text{Blank}(s_2) \wedge \text{Adjacent}(s_1, s_2)$ maka petak apa pun dapat bergerak dalam satu tindakan ke ruang mana pun dan kita mendapatkan heuristik jumlah petak yang salah tempat. Jika kita menghapus $\text{Blank}(s_2)$ maka kita mendapatkan heuristik jarak Manhattan. Mudah untuk melihat bagaimana heuristik ini dapat diturunkan secara otomatis dari deskripsi skema tindakan. Kemudahan memanipulasi skema adalah keuntungan besar dari representasi faktor dari masalah perencanaan, dibandingkan dengan representasi atom dari masalah pencarian.

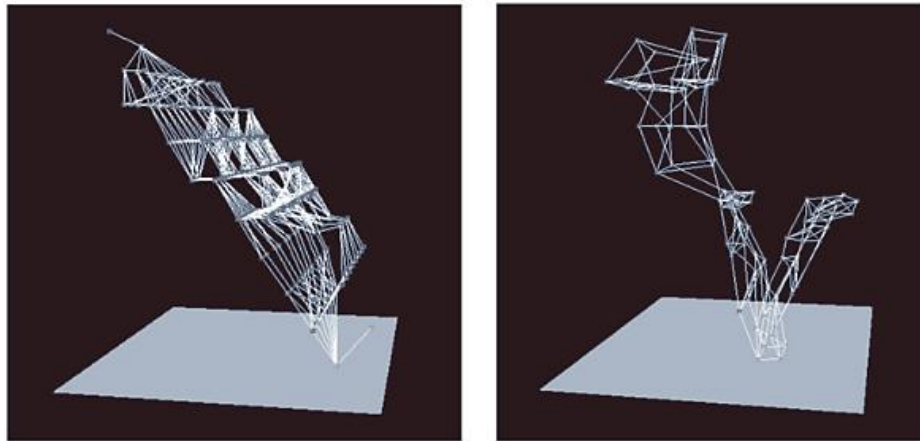
Kemungkinan lain adalah heuristik abaikan hapus daftar. Asumsikan sejenak bahwa semua sasaran dan prasyarat hanya berisi literal positif. Kita ingin membuat versi yang lebih santai dari masalah asli yang akan lebih mudah dipecahkan, dan di mana panjang solusi akan berfungsi sebagai heuristik yang baik. Kita dapat melakukannya dengan menghapus daftar hapus dari semua tindakan (yaitu, menghapus semua literal negatif dari efek). Hal itu memungkinkan untuk membuat kemajuan monotonik menuju sasaran—tidak ada tindakan yang akan membatalkan kemajuan yang dibuat oleh tindakan lain.

Ternyata masih sulit untuk menemukan solusi optimal untuk masalah yang dilonggarkan ini, tetapi solusi perkiraan dapat ditemukan dalam waktu polinomial dengan mendaki bukit. Gambar 1.6 menggambarkan bagian dari ruang keadaan untuk dua masalah perencanaan menggunakan heuristik abaikan-hapus-daftar. Titik-titik mewakili keadaan dan tepinya tindakan, dan tinggi setiap titik di atas bidang bawah mewakili nilai heuristik. Keadaan di bidang bawah adalah solusi. Dalam kedua masalah ini, ada jalur lebar menuju sasaran. Tidak ada jalan buntu, jadi tidak perlu mundur; pencarian mendaki bukit sederhana akan dengan mudah menemukan solusi untuk masalah ini (meskipun itu mungkin bukan solusi optimal).

Masalah yang dilonggarkan memberi kita masalah perencanaan yang disederhanakan—tetapi tetap mahal—hanya untuk menghitung nilai fungsi heuristik. Banyak masalah perencanaan memiliki 10^{100} keadaan atau lebih, dan melonggarkan tindakan tidak mengurangi jumlah keadaan. Oleh karena itu, sekarang kita melihat relaksasi yang mengurangi jumlah status dengan membentuk abstraksi status—pemetaan banyak ke satu dari status dalam representasi dasar masalah ke representasi abstrak.

Bentuk abstraksi status yang paling mudah adalah mengabaikan beberapa fluent. Misalnya, pertimbangkan masalah kargo udara dengan 10 bandara, 50 pesawat, dan 200 buah kargo. Setiap pesawat dapat berada di salah satu dari 10 bandara dan setiap paket dapat berada di salah satu pesawat atau diturunkan di salah satu bandara. Jadi, ada $50^{10} \times 200^{50+10} \approx 10^{155}$ status. Sekarang pertimbangkan masalah tertentu dalam domain tersebut di mana semua paket berada di hanya 5 bandara, dan semua paket di bandara tertentu memiliki tujuan yang sama.

Maka abstraksi yang berguna dari masalah tersebut adalah dengan membuang semua fluent *At* kecuali yang melibatkan satu pesawat dan satu paket di masing-masing dari 5 bandara. Sekarang hanya ada $5^{10} \times 5^{5+10} \approx 10^{17}$ status. Solusi dalam ruang status abstrak ini akan lebih pendek daripada solusi dalam ruang asli (dan dengan demikian akan menjadi heuristik yang dapat diterima), dan solusi abstrak mudah diperluas menjadi solusi untuk masalah asli (dengan menambahkan tindakan *Load* dan *Unload* tambahan).



Gambar 1.6 Dua ruang keadaan dari masalah perencanaan dengan heuristik abaikan-hapus-daftar. Ketinggian di atas bidang bawah adalah skor heuristik dari suatu keadaan; keadaan di bidang bawah adalah tujuan. Tidak ada minimum lokal, jadi pencarian tujuan menjadi mudah. Dari Hoffmann.

Gagasan utama dalam mendefinisikan heuristik adalah dekomposisi: membagi masalah menjadi beberapa bagian, menyelesaikan setiap bagian secara independen, dan kemudian menggabungkan bagian-bagian tersebut. Asumsi independensi subtujuan adalah bahwa biaya penyelesaian konjungsi subtujuan didekati dengan jumlah biaya penyelesaian setiap subtujuan secara independen. Asumsi independensi subtujuan dapat bersifat optimis atau pesimis. Asumsi ini optimis ketika ada interaksi negatif antara subrencana untuk setiap subtujuan—misalnya, ketika tindakan dalam satu subrencana menghapus tujuan yang dicapai oleh subrencana lain. Asumsi ini pesimis, dan karenanya tidak dapat diterima, ketika subrencana berisi tindakan yang berulang—misalnya, dua tindakan yang dapat digantikan oleh satu tindakan dalam rencana gabungan.

Misalkan tujuannya adalah serangkaian fluent G , yang kita bagi menjadi subset terpisah $G_1, \dots, G_n$. Kita kemudian menemukan rencana $P_1, \dots, P_n$ yang menyelesaikan masing-masing subtujuan. Berapa perkiraan biaya rencana untuk mencapai semua G ? Kita dapat menganggap setiap Biaya (P_i) sebagai estimasi heuristik, dan kita tahu bahwa jika kita menggabungkan estimasi dengan mengambil nilai maksimumnya, kita selalu mendapatkan heuristik yang dapat diterima. Jadi, $COST(P_i)$ maksimum dapat diterima, dan terkadang benar-benar tepat: bisa jadi P_i secara kebetulan mencapai semua G_i . Namun, dalam kebanyakan kasus, estimasi tersebut terlalu rendah. Bisakah kita menjumlahkan biayanya? Untuk banyak masalah, itu adalah estimasi yang wajar, tetapi tidak dapat diterima. Kasus terbaik adalah ketika kita dapat menentukan bahwa G_i dan G_j bersifat independen. Jika efek P_i tidak mengubah semua prasyarat dan tujuan P_j , maka estimasi $COST(P_i) + COST(P_j)$ dapat diterima, dan lebih akurat daripada estimasi maksimum. Kami menunjukkan bahwa grafik perencanaan dapat membantu memberikan estimasi heuristik yang lebih baik.

Jelas bahwa ada potensi besar untuk memangkas ruang pencarian dengan membentuk abstraksi. Triknya adalah memilih abstraksi yang tepat dan menggunakannya dengan cara yang

membuat total biaya—mendefinisikan abstraksi, melakukan pencarian abstrak, dan memetakan abstraksi kembali ke masalah awal—lebih rendah daripada biaya untuk menyelesaikan masalah awal. Teknik basis data pola dapat berguna, karena biaya pembuatan basis data pola dapat diamortisasi selama beberapa contoh masalah.

Contoh sistem yang menggunakan heuristik yang efektif adalah *FF*, atau FASTFORWARD, penelusur ruang status maju yang menggunakan heuristik abaikan-hapus-daftar, memperkirakan heuristik dengan bantuan grafik perencanaan. *FF* kemudian menggunakan penelusuran mendaki bukit (dimodifikasi untuk melacak rencana) dengan heuristik untuk menemukan solusi. Ketika mencapai titik puncak atau maksimum lokal—ketika tidak ada tindakan yang mengarah ke kondisi dengan skor heuristik yang lebih baik—maka *FF* menggunakan penelusuran pendalaman berulang hingga menemukan kondisi yang lebih baik, atau menyerah dan memulai kembali pendakian bukit.

1.3 GRAFIK PERENCANAAN

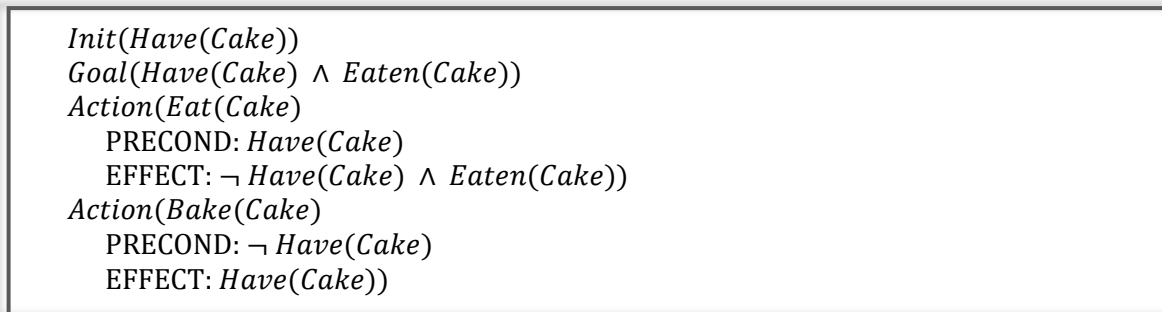
Semua heuristik yang kami sarankan dapat mengalami ketidakakuratan. Bagian ini menunjukkan bagaimana struktur data khusus yang disebut grafik perencanaan dapat digunakan untuk memberikan estimasi heuristik yang lebih baik. Heuristik ini dapat diterapkan pada semua teknik pencarian yang telah kita lihat sejauh ini. Atau, kita dapat mencari solusi pada ruang yang dibentuk oleh grafik perencanaan, menggunakan algoritma yang disebut GRAPHPLAN.

Masalah perencanaan menanyakan apakah kita dapat mencapai status tujuan dari status awal. Misalkan kita diberi pohon dari semua tindakan yang mungkin dari status awal hingga status penerus, dan penerusnya, dan seterusnya. Jika kita mengindeks pohon ini dengan tepat, kita dapat menjawab pertanyaan perencanaan "dapatkah kita mencapai status G dari status S_0 " dengan segera, hanya dengan mencarinya. Tentu saja, pohon tersebut berukuran eksponensial, jadi pendekatan ini tidak praktis. Grafik perencanaan adalah perkiraan ukuran polinomial untuk pohon ini yang dapat dibangun dengan cepat. Grafik perencanaan tidak dapat menjawab secara pasti apakah G dapat dicapai dari S_0 , tetapi dapat memperkirakan berapa banyak langkah yang diperlukan untuk mencapai G . Perkiraan tersebut selalu benar ketika melaporkan bahwa tujuan tidak dapat dicapai, dan tidak pernah melebih-lebihkan jumlah langkah, sehingga merupakan heuristik yang dapat diterima.

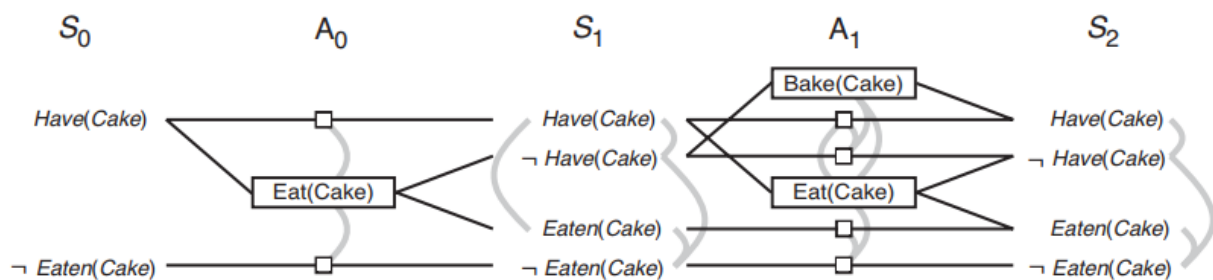
Grafik perencanaan adalah grafik terarah yang diorganisasikan ke dalam beberapa level: pertama level S_0 untuk status awal, yang terdiri dari simpul yang mewakili setiap fasih yang berlaku di S_0 ; kemudian level A_0 yang terdiri dari simpul untuk setiap tindakan dasar yang mungkin berlaku di S_0 ; kemudian level bergantian S_i diikuti oleh A_i ; hingga kita mencapai kondisi terminasi. Secara kasar, S_i berisi semua literal yang dapat berlaku pada waktu i , tergantung pada tindakan yang dieksekusi pada langkah waktu sebelumnya. Jika mungkin P atau $\neg P$ dapat berlaku, maka keduanya akan direpresentasikan dalam S_i .

Secara kasar, A_i berisi semua tindakan yang dapat memenuhi prasyaratnya pada waktu i . Kami katakan "secara kasar" karena grafik perencanaan hanya mencatat sebagian kecil dari kemungkinan interaksi negatif di antara tindakan; oleh karena itu, literal mungkin muncul di

level S_j padahal sebenarnya tidak mungkin benar hingga level berikutnya, jika memang benar. (Literal tidak akan pernah muncul terlambat.) Meskipun ada kemungkinan kesalahan, level j di mana literal pertama kali muncul adalah perkiraan yang baik tentang seberapa sulitnya mencapai literal dari keadaan awal.



Gambar 1.7 Masalah “makan kue dan makan kue juga”.



Gambar 1.8 Grafik perencanaan untuk masalah “makan kue dan makan kue juga” hingga level S_2 . Persegi panjang menunjukkan tindakan (kotak kecil menunjukkan tindakan persistensi), dan garis lurus menunjukkan prasyarat dan efek. Tautan mutex ditampilkan sebagai garis abu-abu melengkung. Tidak semua tautan mutex ditampilkan, karena grafiknya akan terlalu berantakan. Secara umum, jika dua literal adalah mutex di S_i , maka tindakan persistensi untuk literal tersebut akan menjadi mutex di A_i dan kita tidak perlu menggambar tautan mutex tersebut.

Grafik perencanaan hanya berfungsi untuk masalah perencanaan proposisional—yang tidak memiliki variabel. Seperti yang kami sebutkan di halaman 368, mudah untuk memproposisikan sekumpulan skema tindakan. Meskipun ukuran deskripsi masalah meningkat, grafik perencanaan terbukti menjadi alat yang efektif untuk memecahkan masalah perencanaan yang sulit.

Gambar 1.7 menunjukkan masalah perencanaan sederhana, dan Gambar 1.8 menunjukkan grafik perencanaannya. Setiap tindakan di level A_i terhubung ke prasyaratnya di S_i dan efeknya di S_{i+1} . Jadi literal muncul karena suatu tindakan menyebabkannya, tetapi kita juga ingin mengatakan bahwa literal dapat bertahan jika tidak ada tindakan yang meniadakannya. Ini diwakili oleh tindakan persistensi (kadang-kadang disebut *no-op*). Untuk setiap literal C , kita menambahkan tindakan persistensi dengan prasyarat C dan efek C .

Level A_0 pada Gambar 1.8 menunjukkan satu tindakan "nyata", $Eat(Cake)$, bersama dengan dua tindakan persistensi yang digambar sebagai kotak persegi kecil.

Level A_0 berisi semua tindakan yang dapat terjadi dalam status S_0 , tetapi yang lebih penting lagi, level ini mencatat konflik antara tindakan yang akan mencegahnya terjadi bersamaan. Garis abu-abu pada Gambar 1.8 menunjukkan tautan pengecualian bersama (atau mutex). Misalnya, $Eat(Cake)$ saling eksklusif dengan persistensi $Have(Cake)$ atau $\neg Eaten(Cake)$. Kita akan segera melihat bagaimana tautan mutex dihitung.

Level S_1 berisi semua literal yang dapat dihasilkan dari pemilihan subset tindakan apa pun di A_0 , serta tautan mutex (garis abu-abu) yang menunjukkan literal yang tidak dapat muncul bersamaan, terlepas dari pilihan tindakan. Misalnya, $Have(Cake)$ dan $Eaten(Cake)$ adalah mutex:

tergantung pada pilihan tindakan di A_0 , salah satu, tetapi tidak keduanya, bisa jadi hasilnya. Dengan kata lain, S_1 merepresentasikan status keyakinan: sekumpulan status yang mungkin. Anggota dari set ini adalah semua subset dari literal sehingga tidak ada tautan mutex antara anggota subset mana pun. Kita lanjutkan dengan cara ini, bergantian antara level status S_i dan level tindakan S_i hingga kita mencapai titik di mana dua level berurutan identik. Pada titik ini, kita katakan bahwa grafik telah mendatar. Grafik pada Gambar 1.8 mendatar di S_2 .

Yang kita dapatkan adalah struktur di mana setiap level A_i berisi semua tindakan yang berlaku di S_i , beserta kendala yang menyatakan bahwa dua tindakan tidak dapat dieksekusi pada level yang sama. Setiap level S_i berisi semua literal yang dapat dihasilkan dari setiap pilihan tindakan yang mungkin di A_{i-1} , beserta kendala yang menyatakan pasangan literal mana yang tidak mungkin.

Penting untuk dicatat bahwa proses membangun grafik perencanaan tidak memerlukan pemilihan di antara tindakan, yang akan memerlukan pencarian kombinatorial. Sebaliknya, ia hanya mencatat ketidakmungkinan pilihan tertentu menggunakan tautan mutex. Sekarang kita mendefinisikan tautan mutex untuk tindakan dan literal. Relasi mutex berlaku antara dua tindakan pada level tertentu jika salah satu dari tiga kondisi berikut berlaku:

- Efek yang tidak konsisten: satu tindakan meniadakan efek dari yang lain. Misalnya, $Eat(Cake)$ dan persistensi $Have(Cake)$ memiliki efek yang tidak konsisten karena keduanya tidak setuju pada efek $Have(Cake)$.
- Interferensi: salah satu efek dari satu tindakan adalah negasi dari prasyarat yang lain. Misalnya $Eat(Cake)$ mengganggu persistensi $Have(Cake)$ dengan meniadakan prasyaratnya.
- Kebutuhan yang bersaing: salah satu prasyarat dari satu tindakan saling eksklusif dengan prasyarat yang lain. Misalnya, $Bake(Cake)$ dan $Eat(Cake)$ adalah mutex karena keduanya bersaing pada nilai prasyarat $Have(Cake)$.

Relasi mutex berlaku antara dua literal pada level yang sama jika salah satunya adalah negasi dari yang lain atau jika setiap kemungkinan pasangan tindakan yang dapat mencapai dua literal tersebut saling eksklusif. Kondisi ini disebut dukungan tidak konsisten. Misalnya, $Have(Cake)$ dan $Eaten(Cake)$ adalah mutex di S_1 karena satu-satunya cara untuk mencapai $Have(Cake)$, tindakan persistensi, adalah mutex dengan satu-satunya cara untuk mencapai

Eaten(Cake), yaitu *Eat(Cake)*. Di S_2 kedua literal tersebut bukan mutex, karena ada cara baru untuk mencapainya, seperti *Bake(Cake)* dan persistensi *Eaten(Cake)*, yang bukan mutex.

Grafik perencanaan adalah polinomial dalam ukuran masalah perencanaan. Untuk masalah perencanaan dengan l literal dan a tindakan, setiap S_i memiliki tidak lebih dari l node dan l^2 tautan mutex, dan setiap A_i memiliki tidak lebih dari $a + l$ node (termasuk no-ops), $(a + l)^2$ tautan mutex, dan $2(al + l)$ prakondisi dan tautan efek. Dengan demikian, keseluruhan grafik dengan n level memiliki ukuran $O(n(a + l)^2)$. Waktu untuk membangun grafik memiliki kompleksitas yang sama.

Grafik Perencanaan untuk Estimasi Heuristik

Setelah dibuat, grafik perencanaan merupakan sumber informasi yang kaya tentang masalah tersebut. Pertama, jika ada literal tujuan yang gagal muncul di level akhir grafik, maka masalah tersebut tidak dapat dipecahkan. Kedua, kita dapat memperkirakan biaya untuk mencapai literal tujuan g_i dari status s sebagai level di mana g_i pertama kali muncul dalam grafik perencanaan yang dibuat dari status awal s . Kita menyebutnya biaya level g_i . Pada Gambar 1.8, *Have(Cake)* memiliki biaya level 0 dan *Eaten(Cake)* memiliki biaya level 1. Mudah untuk menunjukkan (Latihan 10.10) bahwa estimasi ini dapat diterima untuk tujuan individual. Namun, estimasi tersebut mungkin tidak selalu akurat, karena grafik perencanaan memungkinkan beberapa tindakan di setiap level, sedangkan heuristik hanya menghitung level dan bukan jumlah tindakan. Karena alasan ini, grafik perencanaan serial umum digunakan untuk menghitung heuristik. Grafik serial menegaskan bahwa hanya satu tindakan yang benar-benar dapat terjadi pada setiap langkah waktu tertentu; ini dilakukan dengan menambahkan tautan mutex antara setiap pasangan tindakan nonpersisten. Biaya level yang diekstraksi dari grafik serial sering kali merupakan estimasi biaya aktual yang cukup masuk akal.

Untuk memperkirakan biaya konjungsi tujuan, ada tiga pendekatan sederhana. Heuristik level maksimum hanya mengambil biaya level maksimum dari salah satu tujuan; ini dapat diterima, tetapi belum tentu akurat. Heuristik jumlah level, mengikuti asumsi independensi subtujuan, mengembalikan jumlah biaya level tujuan; ini dapat tidak dapat diterima tetapi berfungsi dengan baik dalam praktik untuk masalah yang sebagian besar dapat diurai. Ini jauh lebih akurat daripada heuristik jumlah tujuan yang tidak terpenuhi dari Bagian 10.2. Untuk masalah kita, estimasi heuristik jumlah-level untuk tujuan konjungtif *Have(Cake) ∧ Eaten(Cake)* akan menjadi $0 + 1 = 1$, sedangkan jawaban yang benar adalah 2, yang dicapai oleh rencana $[Eat(Cake), Bake(Cake)]$. Itu tidak tampak begitu buruk. Kesalahan yang lebih serius adalah jika *Bake(Cake)* tidak ada dalam rangkaian tindakan, maka estimasi akan tetap menjadi 1, padahal sebenarnya tujuan konjungtif tidak mungkin tercapai.

Akhirnya, heuristik set-level menemukan level di mana semua literal dalam tujuan konjungtif muncul dalam grafik perencanaan tanpa ada pasangan yang saling eksklusif. Heuristik ini memberikan nilai yang benar yaitu 2 untuk masalah awal kita dan tak terhingga untuk masalah tanpa *Bake(Cake)*. Itu dapat diterima, mendominasi heuristik level-maksimum, dan bekerja dengan sangat baik pada tugas-tugas yang di dalamnya terdapat

banyak interaksi di antara sub-rencana. Tentu saja, itu tidak sempurna; misalnya, grafik tersebut mengabaikan interaksi antara tiga atau lebih literal.

Sebagai alat untuk menghasilkan heuristik yang akurat, kita dapat melihat grafik perencanaan sebagai masalah yang dilonggarkan yang dapat dipecahkan secara efisien. Untuk memahami sifat masalah yang dilonggarkan, kita perlu memahami dengan tepat apa artinya literal g muncul pada level S_i dalam grafik perencanaan. Idealnya, kita ingin grafik tersebut menjadi jaminan bahwa ada rencana dengan i level tindakan yang mencapai g , dan juga bahwa jika g tidak muncul, tidak ada rencana seperti itu. Sayangnya, membuat jaminan itu sama sulitnya dengan memecahkan masalah perencanaan awal. Jadi grafik perencanaan membuat bagian kedua dari jaminan (jika g tidak muncul, tidak ada rencana), tetapi jika g muncul, maka semua yang dijanjikan grafik perencanaan adalah bahwa ada rencana yang mungkin mencapai g dan tidak memiliki kekurangan yang "jelas".

Kekurangan yang jelas didefinisikan sebagai kekurangan yang dapat dideteksi dengan mempertimbangkan dua tindakan atau dua literal pada satu waktu—dengan kata lain, dengan melihat hubungan mutex. Mungkin ada kekurangan yang lebih halus yang melibatkan tiga, empat, atau lebih tindakan, tetapi pengalaman telah menunjukkan bahwa tidak ada gunanya melakukan upaya komputasi untuk melacak kemungkinan kekurangan ini. Ini mirip dengan pelajaran yang dipelajari dari masalah pemenuhan kendala—bahwa sering kali bermanfaat untuk menghitung konsistensi 2 sebelum mencari solusi, tetapi jarang bermanfaat untuk menghitung konsistensi 3 atau lebih tinggi.

Salah satu contoh masalah yang tidak dapat dipecahkan yang tidak dapat dikenali sebagai masalah tersebut oleh grafik perencanaan adalah masalah blok-dunia di mana tujuannya adalah untuk mendapatkan blok A pada B , B pada C , dan C pada A . Ini adalah tujuan yang mustahil; sebuah menara dengan bagian bawah di atas bagian atas. Namun, grafik perencanaan tidak dapat mendeteksi ketidakmungkinan tersebut, karena dua dari tiga subtujuan dapat dicapai. Tidak ada mutex antara pasangan literal mana pun, hanya antara ketiganya secara keseluruhan. Untuk mendeteksi bahwa masalah ini tidak mungkin, kita harus mencari di atas grafik perencanaan.

Algoritma GRAPHPLAN

Subbagian ini menunjukkan cara mengekstrak rencana langsung dari grafik perencanaan, daripada hanya menggunakan grafik untuk menyediakan heuristik. Algoritma GRAPHPLAN (Gambar 1.9) berulang kali menambahkan level ke grafik perencanaan dengan EXPAND-GRAPH. Setelah semua tujuan muncul sebagai non-mutex dalam grafik, GRAPHPLAN memanggil EXTRACT-SOLUTION untuk mencari rencana yang memecahkan masalah. Jika gagal, ia memperluas level lain dan mencoba lagi, berakhir dengan kegagalan saat tidak ada alasan untuk melanjutkan.

function *GRAPHPLAN*(*problem*) **returns** solution or failure

graph $\leftarrow$ INITIAL – PLANNING – GRAPH(*problem*)

goals $\leftarrow$ CONJUNCTS(*problem*.GOAL)

nogoods $\leftarrow$ an empty hash table

for *tl* = 0 **to** ∞ **do**

if *goals* all non – mutex in S_t of *graph* **then**

solution $\leftarrow$ EXTRACT – SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)

if *solution* $\neq$ failure **then return** *solution*

if *graph* and *nogoods* have both leveled off **then return** failure

graph $\leftarrow$ EXPAND – GRAPH(*graph*, *problem*)

Gambar 1.9 Algoritma GRAPHPLAN. GRAPHPLAN memanggil EXPAND-GRAPH untuk menambahkan level hingga solusi ditemukan oleh EXTRACT-SOLUTION, atau tidak ada solusi yang memungkinkan.

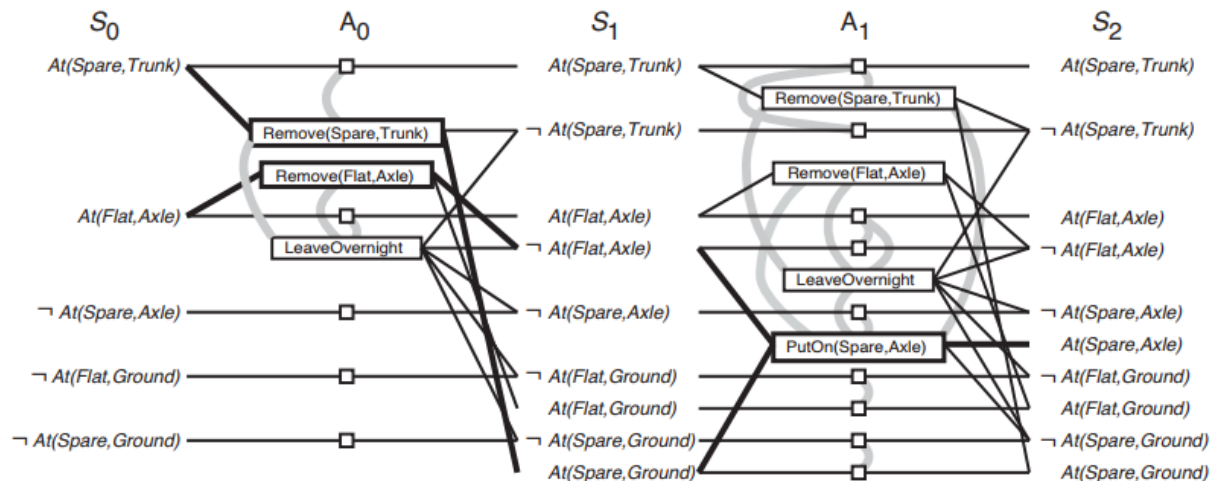
Mari kita telusuri operasi GRAPHPLAN pada masalah ban serep. Grafik ditunjukkan pada Gambar 1.10. Baris pertama GRAPHPLAN menginisialisasi grafik perencanaan ke grafik satu level (S_0) yang merepresentasikan status awal. Fluent positif dari status awal deskripsi masalah ditunjukkan, seperti halnya fluent negatif yang relevan. Yang tidak ditunjukkan adalah literal positif yang tidak berubah (seperti *Tire(Spare)*) dan literal negatif yang tidak relevan.

Sasaran *At(Spare, Axle)* tidak ada di S_0 , jadi kita tidak perlu memanggil EXTRACT-SOLUTION — kita yakin bahwa belum ada solusi. Sebaliknya, EXPAND-GRAPH menambahkan ke A_0 tiga tindakan yang prasyaratnya ada di level S_0 (yaitu, semua tindakan kecuali *PutOn(Spare, Axle)*), bersama dengan tindakan persistensi untuk semua literal di S_0 . Efek dari tindakan ditambahkan di level S_1 . EXPAND-GRAPH kemudian mencari relasi mutex dan menambahkannya ke grafik.

At(Spare, Axle) masih belum ada di S_1 , jadi sekali lagi kita tidak memanggil EXTRACT-SOLUTION. Kita memanggil EXPAND-GRAPH lagi, menambahkan A_1 dan S_1 dan memberi kita grafik perencanaan yang ditunjukkan pada Gambar 1.10. Sekarang setelah kita memiliki tindakan yang lengkap, ada baiknya untuk melihat beberapa contoh relasi mutex dan penyebabnya:

- ❖ Efek yang tidak konsisten: *Remove(Spare, Trunk)* adalah mutex dengan *LeaveOvernight* karena yang satu memiliki efek *At(Spare, Ground)* dan yang lainnya memiliki negasinya.

❖



Gambar 1.10 Grafik perencanaan untuk masalah ban serep setelah perluasan ke level S_2 . Tautan mutex ditampilkan sebagai garis abu-abu. Tidak semua tautan ditampilkan, karena grafik akan terlalu berantakan jika kita menampilkan semuanya. Solusinya ditunjukkan dengan garis tebal dan garis luar.

- ❖ Interferensi: *Remove (Flat, Axle)* adalah mutex dengan *LeaveOvernight* karena yang satu memiliki prasyarat *At(Flat, Axle)* dan yang lainnya memiliki negasinya sebagai efek.
- ❖ Kebutuhan yang bersaing: *PutOn(Spare, Axle)* adalah mutex dengan *Remove (Flat, Axle)* karena yang satu memiliki *At(Flat, Axle)* sebagai prasyarat dan yang lainnya memiliki negasinya.
- ❖ Dukungan yang tidak konsisten: *At(Spare, Axle)* adalah mutex dengan *At(Flat, Axle)* di S_2 karena satu-satunya cara untuk mencapai *At(Spare, Axle)* adalah dengan *PutOn(Spare, Axle)*, dan itu adalah mutex dengan tindakan persistensi yang merupakan satu-satunya cara untuk mencapai *At(Flat, Axle)*. Dengan demikian, relasi mutex mendeteksi konflik langsung yang muncul dari upaya untuk menempatkan dua objek di tempat yang sama pada waktu yang sama.

Kali ini, ketika kita kembali ke awal loop, semua literal dari tujuan hadir di S_2 , dan tidak ada satu pun yang mutex dengan yang lain. Itu berarti bahwa solusi mungkin ada, dan EXTRACT-SOLUTION akan mencoba menemukannya. Kita dapat merumuskan EXTRACT-SOLUTION sebagai masalah pemenuhan kendala Boolean (CSP) di mana variabelnya adalah tindakan di setiap level, nilai untuk setiap variabel ada di dalam atau di luar rencana, dan kendalanya adalah mutex dan kebutuhan untuk memenuhi setiap tujuan dan prasyarat. Atau, kita dapat mendefinisikan EXTRACT-SOLUTION sebagai masalah pencarian mundur, di mana setiap status dalam pencarian berisi penunjuk ke level dalam grafik perencanaan dan serangkaian tujuan yang tidak terpenuhi. Kita mendefinisikan masalah pencarian ini sebagai berikut:

- Status awal adalah level terakhir dari grafik perencanaan, S_n , bersama dengan serangkaian tujuan dari masalah perencanaan.
- Tindakan yang tersedia dalam status pada level S_i adalah memilih setiap subset tindakan bebas konflik di A_{i-1} yang efeknya mencakup tujuan dalam status tersebut.

Status yang dihasilkan memiliki level S_{i-1} dan memiliki serangkaian tujuannya sebagai prasyarat untuk serangkaian tindakan yang dipilih. Dengan "bebas konflik", yang kami maksud adalah serangkaian tindakan sedemikian rupa sehingga tidak ada dua di antaranya yang mutex dan tidak ada dua prasyaratnya yang mutex.

- Tujuannya adalah mencapai status pada level S_0 sedemikian rupa sehingga semua tujuan terpenuhi.
- Biaya setiap tindakan adalah 1.

Untuk masalah khusus ini, kami mulai di S_2 dengan tujuan $At(Spare, Axle)$. Satu-satunya pilihan yang kami miliki untuk mencapai serangkaian tujuan adalah $PutOn(Spare, Axle)$. Itu membawa kita ke status pencarian di S_1 dengan tujuan $At(Spare, Ground)$ dan $\neg At(Flat, Axle)$. Yang pertama hanya dapat dicapai dengan $Remove(Spare, Trunk)$, dan yang kedua dengan $Remove(Flat, Axle)$ atau $LeaveOvernight$. Namun $LeaveOvernight$ adalah mutex dengan $Remove(Spare, Trunk)$, jadi satu-satunya solusi adalah memilih $Remove(Spare, Trunk)$ dan $Remove(Flat, Axle)$. Itu membawa kita ke status pencarian di S_0 dengan tujuan $At(Spare, Trunk)$ dan $At(Flat, Axle)$. Keduanya hadir dalam status tersebut, jadi kita memiliki solusi: tindakan $Remove(Spare, Trunk)$ dan $Remove(Flat, Axle)$ di level A_0 , diikuti oleh $PutOn(Spare, Axle)$ di A_1 .

Dalam kasus di mana EXTRACT-SOLUTION gagal menemukan solusi untuk serangkaian tujuan pada level tertentu, kita mencatat pasangan $(level, goals)$ sebagai no-good, sama seperti yang kita lakukan dalam constraint learning untuk CSP. Setiap kali EXTRACT-SOLUTION dipanggil lagi dengan level dan tujuan yang sama, kita dapat menemukan no-good yang terekam dan segera mengembalikan kegagalan daripada mencari lagi. Kita segera melihat bahwa no-good juga digunakan dalam uji penghentian.

Kita tahu bahwa perencanaan bersifat PSPACE-lengkap dan bahwa membangun grafik perencanaan membutuhkan waktu polinomial, jadi ekstraksi solusi pasti sulit dilakukan dalam kasus terburuk. Oleh karena itu, kita memerlukan beberapa panduan heuristik untuk memilih di antara tindakan selama pencarian mundur. Satu pendekatan yang bekerja dengan baik dalam praktik adalah algoritma greedy berdasarkan biaya level literal. Untuk serangkaian tujuan apa pun, kita melanjutkan dalam urutan berikut:

1. Pilih literal dengan biaya level tertinggi terlebih dahulu.
2. Untuk mencapai literal itu, pilih tindakan dengan prasyarat yang lebih mudah. Yaitu, pilih tindakan sedemikian rupa sehingga jumlah (atau maksimum) biaya level prasyaratnya adalah yang terkecil.

Penghentian GRAPHPLAN

Sejauh ini, kita telah melewati pertanyaan tentang penghentian. Di sini kita menunjukkan bahwa GRAPHPLAN pada kenyataannya akan berhenti dan mengembalikan kegagalan ketika tidak ada solusi.

Hal pertama yang harus dipahami adalah mengapa kita tidak dapat berhenti mengembangkan grafik segera setelah grafik tersebut mendarat. Pertimbangkan domain kargo udara dengan satu pesawat dan n buah kargo di bandara A , yang semuanya memiliki bandara B sebagai tujuan mereka. Dalam versi masalah ini, hanya satu buah kargo yang dapat dimuat

di dalam pesawat pada satu waktu. Grafik akan mendatar pada level 4, yang mencerminkan fakta bahwa untuk setiap buah kargo, kita dapat memuatnya, menerbangkannya, dan membongkarnya di tempat tujuan dalam tiga langkah.

Namun itu tidak berarti bahwa solusi dapat diekstraksi dari grafik pada level 4; pada kenyataannya solusi akan membutuhkan $4n - 1$ langkah: untuk setiap buah kargo yang kita muat, terbangkan, dan bongkar, dan untuk semua kecuali buah terakhir kita perlu terbang kembali ke bandara A untuk mendapatkan buah berikutnya. Berapa lama kita harus terus mengembangkan setelah grafik mendatar?

Jika fungsi EXTRACT-SOLUTION gagal menemukan solusi, maka pasti ada setidaknya satu set tujuan yang tidak dapat dicapai dan ditandai sebagai tidak baik. Jadi jika mungkin ada lebih sedikit yang tidak baik di level berikutnya, maka kita harus melanjutkan. Begitu grafik itu sendiri dan yang tidak baik sama-sama mendatar, tanpa solusi yang ditemukan, kita dapat mengakhiri dengan kegagalan karena tidak ada kemungkinan perubahan berikutnya yang dapat menambahkan solusi.

Sekarang yang harus kita lakukan adalah membuktikan bahwa grafik dan no-goods akan selalu mendatar. Kunci pembuktian ini adalah bahwa properti tertentu dari grafik perencanaan adalah meningkat atau menurun secara monoton. " X meningkat secara monoton" berarti bahwa himpunan X pada level $i + 1$ adalah superset (tidak harus tepat) dari himpunan pada level i . Properti-propertinya adalah sebagai berikut:

- Literal meningkat secara monoton: Setelah literal muncul pada level tertentu, literal tersebut akan muncul pada semua level berikutnya. Ini karena tindakan persistensi; setelah literal muncul, tindakan persistensi menyebabkannya tetap ada selamanya.
- Tindakan meningkat secara monoton: Setelah tindakan muncul pada level tertentu, literal tersebut akan muncul pada semua level berikutnya. Ini adalah konsekuensi dari peningkatan literal yang monoton; jika prasyarat suatu tindakan muncul pada satu level, prasyarat tersebut akan muncul pada level berikutnya, dan dengan demikian tindakan tersebut juga akan muncul.
- Mutex menurun secara monoton: Jika dua tindakan bersifat mutex pada level A_i tertentu, maka tindakan tersebut juga akan bersifat mutex untuk semua level sebelumnya di mana keduanya muncul. Hal yang sama berlaku untuk mutex di antara literal. Mungkin tidak selalu tampak seperti itu pada gambar, karena gambar tersebut memiliki penyederhanaan: gambar tersebut tidak menampilkan literal yang tidak dapat berada pada level S_i maupun tindakan yang tidak dapat dijalankan pada level A_i . Kita dapat melihat bahwa "mutex menurun secara monoton" adalah benar jika Anda menganggap bahwa literal dan tindakan yang tidak terlihat ini bersifat mutex dengan segala hal. Pembuktiannya dapat ditangani berdasarkan kasus: jika tindakan A dan B bersifat mutex pada level A_i , hal itu pasti karena salah satu dari tiga jenis mutex. Dua yang pertama, efek yang tidak konsisten dan interferensi, adalah properti dari tindakan itu sendiri, jadi jika tindakan tersebut bersifat mutex pada A_i , tindakan tersebut akan bersifat mutex pada setiap level. Kasus ketiga, kebutuhan yang bersaing, bergantung pada kondisi di level S_i : level

tersebut harus berisi prasyarat A yang mutex dengan prasyarat B . Nah, kedua prasyarat ini bisa mutex jika keduanya merupakan negasi satu sama lain (yang mana keduanya akan mutex di setiap level) atau jika semua tindakan untuk mencapai satu hal adalah mutex dengan semua tindakan untuk mencapai hal lainnya. Namun, kita sudah tahu bahwa tindakan yang tersedia meningkat secara monoton, jadi, dengan induksi, mutex pasti menurun.

- Hal-hal yang tidak berguna menurun secara monoton: Jika serangkaian tujuan tidak dapat dicapai pada level tertentu, maka tujuan tersebut tidak dapat dicapai pada level sebelumnya. Buktinya adalah dengan kontradiksi: jika tujuan tersebut dapat dicapai pada beberapa level sebelumnya, maka kita dapat menambahkan tindakan persistensi untuk membuatnya dapat dicapai pada level berikutnya.

Year	Track	Winning Systems (approaches)
2008	Optimal	GAMER (model checking, bidirectional search)
2008	Satisficing	LAMA (fast downward search with FF heuristic)
2006	Optimal	SATPLAN, MAXPLAN (Boolean satisfiability)
2006	Satisficing	SGPLAN (forward search; partitions into independent subproblems)
2004	Optimal	SATPLAN (Boolean satisfiability)
2004	Satisficing	FAST DIAGONALLY DOWNWARD (forward search with causal graph)
2002	Automated	LPG (local search, planning graphs converted to CSPs)
2002	Hand-coded	TLPLAN (temporal action logic with control rules for forward search)
2000	Automated	FF (forward search)
2000	Hand-coded	TALPLANNER (temporal action logic with control rules for forward search)
1998	Automated	IPP (planning graphs); HSP (forward search)

Gambar 1.11 Beberapa sistem dengan kinerja terbaik dalam Kompetisi Perencanaan Internasional. Setiap tahun ada berbagai jalur: “Optimal” berarti para perencana harus menghasilkan rencana sesingkat mungkin, sementara “Satisficing” berarti solusi yang tidak optimal diterima. “Hand-coded” berarti heuristik khusus domain diperbolehkan; “Automated” berarti tidak diperbolehkan.

Karena tindakan dan literal meningkat secara monoton dan karena hanya ada sejumlah tindakan dan literal yang terbatas, pasti ada level yang memiliki jumlah tindakan dan literal yang sama dengan level sebelumnya. Karena mutex dan no-goods berkurang, dan karena tidak akan pernah ada lebih sedikit dari nol mutex atau no-goods, pasti ada level yang memiliki jumlah mutex dan no-goods yang sama dengan level sebelumnya. Setelah grafik mencapai status ini, jika salah satu tujuan hilang atau mutex dengan tujuan lain, maka kita dapat menghentikan algoritma GRAPHPLAN dan mengembalikan kegagalan. Itu menyimpulkan sketsa pembuktian.

1.4 PENDEKATAN PERENCANAAN KLASIK LAINNYA

Saat ini pendekatan yang paling populer dan efektif untuk perencanaan yang sepenuhnya otomatis adalah:

- Penerjemahan ke masalah kepuasan Boolean (SAT)
- Pencarian ruang status maju dengan heuristik yang dibuat dengan saksama

- Pencarian menggunakan grafik perencanaan

Ketiga pendekatan ini bukan satu-satunya yang dicoba dalam sejarah perencanaan otomatis selama 40 tahun. Gambar 1.11 menunjukkan beberapa sistem teratas dalam Kompetisi Perencanaan Internasional, yang telah diadakan setiap tahun genap sejak 1998. Di bagian ini, pertama-tama kami menjelaskan penerjemahan ke masalah kepuasan dan kemudian menjelaskan tiga pendekatan berpengaruh lainnya: perencanaan sebagai deduksi logis orde pertama; sebagai pemenuhan kendala; dan sebagai penyempurnaan rencana.

Perencanaan Klasik Sebagai Kepuasan Boolean

Kita melihat bagaimana SATPLAN memecahkan masalah perencanaan yang dinyatakan dalam logika proporsional. Di sini kita menunjukkan cara menerjemahkan deskripsi PDDL ke dalam bentuk yang dapat diproses oleh SATPLAN. Terjemahan tersebut merupakan serangkaian langkah langsung:

- Proposisikan tindakan: ganti setiap skema tindakan dengan serangkaian tindakan dasar yang dibentuk dengan mensubstitusikan konstanta untuk setiap variabel. Tindakan dasar ini bukan bagian dari terjemahan, tetapi akan digunakan dalam langkah-langkah berikutnya.
- Tentukan status awal: nyatakan F^0 untuk setiap fluent F dalam status awal masalah, dan $\neg F$ untuk setiap fluent yang tidak disebutkan dalam status awal.
- Proposisikan tujuan: untuk setiap variabel dalam tujuan, ganti literal yang memuat variabel dengan disjungsi atas konstanta. Misalnya, tujuan memiliki blok A pada blok lain, $On(A, x) \wedge Block(x)$ di dunia dengan objek A, B dan C , akan digantikan dengan tujuan

$$(On(A, A) \wedge Block(A)) \vee (On(A, B) \wedge Block(B)) \vee (On(A, C) \wedge Block(C)).$$

- Tambahkan aksioma status penerus: Untuk setiap F fasih, tambahkan aksioma dalam bentuk

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t),$$

di mana $ActionCausesF$ adalah disjungsi dari semua tindakan dasar yang memiliki F dalam daftar tambah, dan $ActionCausesNotF$ adalah disjungsi dari semua tindakan dasar yang memiliki F dalam daftar hapus.

- Tambahkan aksioma prasyarat: Untuk setiap tindakan dasar A , tambahkan aksioma $A^t \Rightarrow PRE(A)^t$, yaitu, jika suatu tindakan dilakukan pada waktu t , maka prasyaratnya pasti benar.
- Tambahkan aksioma pengecualian tindakan: katakanlah bahwa setiap tindakan berbeda dari setiap tindakan lainnya. Translasi yang dihasilkan adalah dalam bentuk yang dapat kita berikan kepada SATPLAN untuk menemukan solusi.

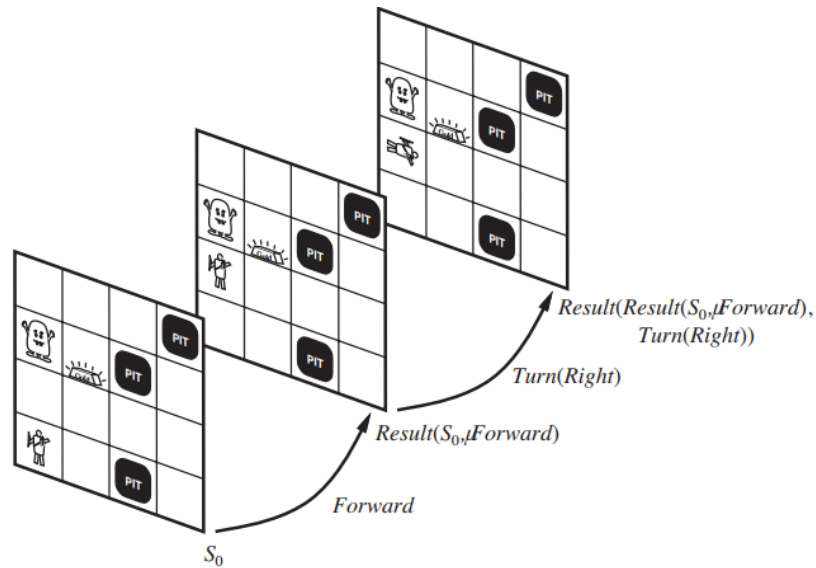
Perencanaan Sebagai Deduksi Logis Orde Pertama: Kalkulus Situasi

PDDL adalah bahasa yang dengan hati-hati menyeimbangkan ekspresi bahasa dengan kompleksitas algoritme yang beroperasi di dalamnya. Namun, beberapa masalah tetap sulit diungkapkan dalam PDDL. Misalnya, kita tidak dapat menyatakan tujuan "memindahkan semua kargo dari A ke B terlepas dari berapa banyak kargo yang ada" dalam PDDL, tetapi kita dapat melakukannya dalam logika orde pertama, menggunakan kuantifier universal. Demikian pula, logika orde pertama dapat secara ringkas menyatakan batasan global seperti "tidak lebih dari empat robot dapat berada di tempat yang sama pada waktu yang sama." PDDL hanya dapat mengatakan ini dengan prasyarat berulang pada setiap tindakan yang mungkin melibatkan pemindahan.

Representasi logika proposisional dari masalah perencanaan juga memiliki keterbatasan, seperti fakta bahwa gagasan waktu dikaitkan langsung dengan fasih. Misalnya, *South*² berarti "agen menghadap ke selatan pada waktu 2." Dengan representasi itu, tidak ada cara untuk mengatakan "agen akan menghadap ke selatan pada waktu 2 jika melakukan belokan kanan pada waktu 1; jika tidak, agen akan menghadap ke timur." Logika orde pertama memungkinkan kita mengatasi keterbatasan ini dengan mengganti gagasan waktu linear dengan gagasan situasi percabangan, menggunakan representasi yang disebut kalkulus situasi yang bekerja seperti ini:

- Keadaan awal disebut situasi. Jika s adalah situasi dan a adalah tindakan, maka $\text{RESULT}(s, a)$ juga merupakan situasi. Tidak ada situasi lain. Jadi, situasi berhubungan dengan urutan, atau riwayat, tindakan. Anda juga dapat menganggap situasi sebagai hasil penerapan tindakan, tetapi perhatikan bahwa dua situasi sama hanya jika awal dan tindakannya sama: $(\text{RESULT}(s, a) = \text{RESULT}(s', a')) \Leftrightarrow (s = s' \wedge a = a')$. Beberapa contoh tindakan dan situasi ditunjukkan pada Gambar 1.12.
- Fungsi atau relasi yang dapat bervariasi dari satu situasi ke situasi berikutnya adalah fasih. Berdasarkan konvensi, situasi s selalu menjadi argumen terakhir untuk fluent, misalnya $\text{At}(x, l, s)$ adalah fluent relasional yang bernilai benar ketika objek x berada di lokasi l dalam situasi s , dan *Location* adalah fluent fungsional sehingga $\text{Location}(x, s) = l$ berlaku dalam situasi yang sama dengan $\text{At}(x, l, s)$.
- Setiap prasyarat tindakan dijelaskan dengan aksioma kemungkinan yang menyatakan kapan tindakan dapat dilakukan. Aksioma ini memiliki bentuk $\Phi(s) \Rightarrow \text{Poss}(a, s)$ di mana $\Phi(s)$ adalah beberapa rumus yang melibatkan s yang menjelaskan prasyarat tersebut. Contoh dari dunia wumpus menyatakan bahwa adalah mungkin untuk menembak jika agen tersebut hidup dan memiliki anak panah:

$$\text{Alive}(\text{Agent}, s) \wedge \text{Have}(\text{Agent}, \text{Arrow}, s) \Rightarrow \text{Poss}(\text{Shoot}, s)$$



Gambar 1.12 Situasi sebagai hasil tindakan di dunia wumpus.

- Setiap fluent dijelaskan dengan aksioma status penerus yang menyatakan apa yang terjadi pada fluent tersebut, tergantung pada tindakan apa yang diambil. Ini mirip dengan pendekatan yang kami ambil untuk logika proposisional. Aksioma tersebut memiliki bentuk

$$\begin{aligned} \text{Action is possible} &\Rightarrow \\ (\text{Fluent is true in result state} &\Leftrightarrow \text{Action's effect made it true} \\ &\vee \text{It was true before and action left it alone}). \end{aligned}$$

Misalnya, aksioma untuk Holding yang fasih secara relasional menyatakan bahwa agen sedang memegang sejumlah emas g setelah menjalankan aksi yang mungkin jika dan hanya jika aksi tersebut adalah *Grab* of g atau jika agen sudah memegang g dan aksi tersebut tidak melepaskannya:

$$\begin{aligned} \text{Poss}(a, s) &\Rightarrow \\ (\text{Holding}(\text{Agent}, g, \text{Result}(a, s)) &\Leftrightarrow \\ a = \text{Grab}(g) \vee (\text{Holding}(\text{Agent}, g, s) &\wedge a = \text{Release}(g))). \end{aligned}$$

- Kita memerlukan aksioma tindakan yang unik sehingga agen dapat menyimpulkan bahwa, misalnya, $a \neq \text{Release}(g)$. Untuk setiap pasangan nama tindakan A_i dan A_j yang berbeda, kita memiliki aksioma yang menyatakan bahwa tindakannya berbeda:

$$A_i(x, \dots) \neq A_j(y, \dots)$$

dan untuk setiap nama tindakan A_i kita memiliki aksioma yang mengatakan dua penggunaan nama tindakan itu sama jika dan hanya jika semua argumennya sama:

$$A_i(x_1, \dots, x_n) = A_i(y_1, \dots, y_n) \Leftrightarrow x_1 = y_1 \wedge \dots \wedge x_n = y_n$$

- Solusi adalah situasi (dan karenanya serangkaian tindakan) yang memenuhi tujuan.

Pekerjaan dalam kalkulus situasi telah banyak membantu mendefinisikan semantik formal perencanaan dan membuka area investigasi baru. Namun sejauh ini belum ada program perencanaan skala besar yang praktis berdasarkan deduksi logis atas kalkulus situasi. Hal ini sebagian karena kesulitan melakukan inferensi yang efisien dalam FOL, tetapi terutama karena bidang tersebut belum mengembangkan heuristik yang efektif untuk perencanaan dengan kalkulus situasi.

Perencanaan Sebagai Pemenuhan Kendala

Kita telah melihat bahwa pemenuhan kendala memiliki banyak kesamaan dengan pemenuhan Boolean, dan kita telah melihat bahwa teknik CSP efektif untuk masalah penjadwalan, jadi tidak mengherankan bahwa adalah mungkin untuk mengodekan masalah perencanaan terbatas (yaitu, masalah menemukan rencana dengan panjang k) sebagai masalah pemenuhan kendala (CSP).

Pengodean ini mirip dengan pengodean untuk masalah SAT, dengan satu penyederhanaan penting: pada setiap langkah waktu, kita hanya memerlukan satu variabel, $Action^t$, yang domainnya adalah kumpulan tindakan yang mungkin. Kita tidak lagi memerlukan satu variabel untuk setiap tindakan, dan kita tidak memerlukan aksioma pengecualian tindakan. Dimungkinkan juga untuk mengodekan grafik perencanaan ke dalam CSP. Ini adalah pendekatan yang diambil oleh GP-CSP.

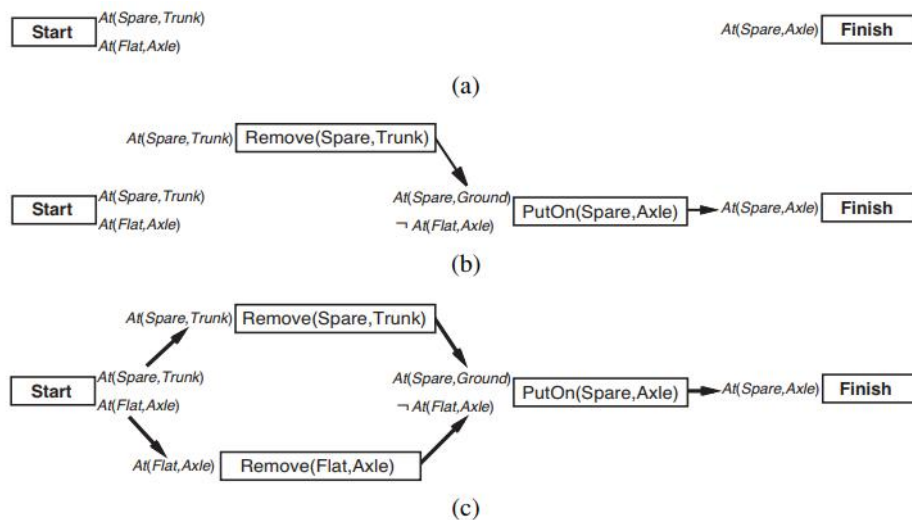
Perencanaan Sebagai Penyempurnaan Rencana yang Diurutkan Sebagian

Semua pendekatan yang telah kita lihat sejauh ini menyusun rencana yang diurutkan secara total yang terdiri dari urutan tindakan yang benar-benar linier. Representasi ini mengabaikan fakta bahwa banyak submasalah yang independen. Solusi untuk masalah kargo udara terdiri dari urutan tindakan yang benar-benar diurutkan, namun jika 30 paket sedang dimuat ke satu pesawat di satu bandara dan 50 paket sedang dimuat ke pesawat lain di bandara lain, tampaknya tidak ada gunanya untuk menghasilkan urutan linier yang ketat dari 80 tindakan pemuatan; dua subset tindakan harus dipikirkan secara independen.

Alternatifnya adalah dengan merepresentasikan rencana sebagai struktur yang diurutkan sebagian: rencana adalah serangkaian tindakan dan serangkaian kendala dalam bentuk $Before(a_i, a_j)$ yang menyatakan bahwa satu tindakan terjadi sebelum tindakan lainnya. Di bagian bawah Gambar 1.13, kita melihat rencana yang diurutkan sebagian yang merupakan solusi untuk masalah ban serep. Tindakan adalah kotak dan kendala pengurutan adalah anak panah. Perhatikan bahwa $Remove(Spare, Trunk)$ dan $Remove(Flat, Axle)$ dapat dilakukan dalam urutan mana pun asalkan keduanya diselesaikan sebelum tindakan $PutOn(Spare, Axle)$.

Rencana yang diurutkan sebagian dibuat dengan pencarian melalui ruang rencana, bukan melalui ruang status. Kita mulai dengan rencana kosong yang hanya terdiri dari status

awal dan tujuan, tanpa tindakan di antaranya, seperti di bagian atas Gambar 1.13. Prosedur pencarian kemudian mencari cacat dalam rencana, dan membuat tambahan pada rencana untuk memperbaiki cacat (atau jika tidak ada perbaikan yang dapat dilakukan, pencarian mundur dan mencoba sesuatu yang lain). Cacat adalah apa pun yang membuat rencana parsial menjadi solusi. Misalnya, satu cacat dalam rencana kosong adalah tidak ada tindakan yang tercapai $At(Spare, Axle)$. Satu cara untuk memperbaiki cacat adalah dengan memasukkan ke dalam rencana tindakan $PutOn(Spare, Axle)$.



Gambar 1.13 (a) masalah ban dinyatakan sebagai rencana kosong. (b) rencana yang belum selesai dan belum diurutkan sebagian untuk masalah ban. Kotak mewakili tindakan dan anak panah menunjukkan bahwa satu tindakan harus terjadi sebelum tindakan lainnya. (c) solusi yang sudah selesai dan belum diurutkan sebagian.

Tentu saja itu memperkenalkan beberapa cacat baru: prasyarat tindakan baru tidak tercapai. Pencarian terus menambah rencana (mundur jika perlu) hingga semua cacat teratasi, seperti di bagian bawah Gambar 1.13. Di setiap langkah, kami membuat komitmen sesedikit mungkin untuk memperbaiki cacat. Misalnya, dalam menambahkan tindakan $Remove(Spare, Trunk)$ kami perlu berkomitmen untuk membuatnya terjadi sebelum $PutOn(Spare, Axle)$, tetapi kami tidak membuat komitmen lain yang menempatkannya sebelum atau setelah tindakan lain. Jika ada variabel dalam skema tindakan yang dapat dibiarkan tidak terikat, kami akan melakukannya.

Pada tahun 1980an dan 90an, perencanaan urutan parsial dipandang sebagai cara terbaik untuk menangani masalah perencanaan dengan submasalah independen—bagaimanapun juga, itu adalah satu-satunya pendekatan yang secara eksplisit mewakili cabang-cabang independen dari suatu rencana. Di sisi lain, pendekatan ini memiliki kelemahan karena tidak memiliki representasi eksplisit dari keadaan dalam model transisi keadaan. Hal itu membuat beberapa perhitungan menjadi rumit. Pada tahun 2000, perencanaan pencarian ke depan telah mengembangkan heuristik yang sangat baik yang memungkinkan mereka untuk secara efisien menemukan submasalah independen yang dirancang untuk perencanaan urutan parsial. Akibatnya, perencanaan urutan parsial tidak kompetitif pada masalah

perencanaan klasik yang sepenuhnya otomatis. Namun, perencanaan urutan parsial tetap menjadi bagian penting dari bidang ini. Untuk beberapa tugas tertentu, seperti penjadwalan operasi, perencanaan urutan parsial dengan heuristik spesifik domain adalah teknologi pilihan. Banyak dari sistem ini menggunakan pustaka rencana tingkat tinggi. Perencanaan dengan urutan parsial juga sering digunakan dalam domain yang mengharuskan manusia untuk memahami rencana tersebut. Rencana operasional untuk wahana antariksa dan penjelajah Mars dibuat oleh perencana dengan urutan parsial dan kemudian diperiksa oleh operator manusia sebelum diunggah ke wahana untuk dieksekusi. Pendekatan penyempurnaan rencana memudahkan manusia untuk memahami apa yang dilakukan algoritma perencanaan dan memverifikasi bahwa rencana tersebut benar.

1.5 ANALISIS PENDEKATAN PERENCANAAN

Perencanaan menggabungkan dua area utama AI yang telah kita bahas sejauh ini: pencarian dan logika. Perencana dapat dilihat sebagai program yang mencari solusi atau sebagai program yang (secara konstruktif) membuktikan keberadaan solusi. Persilangan ide dari kedua area tersebut telah menghasilkan peningkatan kinerja yang mencapai beberapa kali lipat dalam dekade terakhir dan peningkatan penggunaan perencana dalam aplikasi industri. Sayangnya, kita belum memiliki pemahaman yang jelas tentang teknik mana yang paling berhasil untuk jenis masalah apa. Sangat mungkin, teknik baru akan muncul yang mendominasi metode yang ada.

Perencanaan terutama merupakan latihan dalam mengendalikan ledakan kombinatorial. Jika ada n proposisi dalam suatu domain, maka ada 2^n keadaan. Seperti yang telah kita lihat, perencanaan sulit dilakukan. Melawan pesimisme seperti itu, identifikasi submasalah independen dapat menjadi senjata yang ampuh. Dalam kasus terbaik—dekomposisi penuh dari masalah—kita mendapatkan percepatan eksponensial. Namun, dekomposisi dihancurkan oleh interaksi negatif antara tindakan. GRAPHPLAN merekam mutex untuk menunjukkan di mana interaksi yang sulit berada. SATPLAN merepresentasikan rentang relasi mutex yang serupa, tetapi melakukannya dengan menggunakan bentuk CNF umum daripada struktur data tertentu. Pencarian maju mengatasi masalah secara heuristik dengan mencoba menemukan pola (subset proposisi) yang mencakup submasalah independen. Karena pendekatan ini heuristik, pendekatan ini dapat bekerja bahkan ketika submasalah tidak sepenuhnya independen. Terkadang dimungkinkan untuk memecahkan masalah secara efisien dengan mengenali bahwa interaksi negatif dapat dikesampingkan. Kami mengatakan bahwa suatu masalah memiliki subtujuan yang dapat diserialkan jika ada urutan subtujuan sedemikian rupa sehingga perencana dapat mencapainya dalam urutan itu tanpa harus membatalkan salah satu subtujuan yang dicapai sebelumnya.

Misalnya, di dunia blok, jika tujuannya adalah membangun menara (misalnya, A di B , yang selanjutnya ada di C , yang selanjutnya ada di *Table*, maka subtujuannya dapat diserialisasikan dari bawah ke atas: jika kita pertama kali mencapai C di *Table*, kita tidak perlu membatalkannya saat kita mencapai subtujuan lainnya. Perencana yang menggunakan trik

dari bawah ke atas dapat memecahkan masalah apa pun di dunia blok tanpa melakukan penelusuran balik (meskipun mungkin tidak selalu menemukan rencana terpendek).

Sebagai contoh yang lebih rumit, untuk perencana Agen Jarak Jauh yang memimpin wahana antariksa Deep Space One milik NASA, ditentukan bahwa proposisi yang terlibat dalam memimpin wahana antariksa dapat diserialisasikan. Ini mungkin tidak terlalu mengejutkan, karena wahana antariksa dirancang oleh para insinyurnya agar semudah mungkin dikendalikan (tunduk pada kendala lain). Dengan memanfaatkan urutan tujuan yang diserialisasikan, perencana Agen Jarak Jauh dapat menghilangkan sebagian besar pencarian. Ini berarti bahwa ia cukup cepat untuk mengendalikan wahana antariksa secara real time, sesuatu yang sebelumnya dianggap mustahil.

Perencana seperti GRAPHPLAN, SATPLAN, dan FF telah memajukan bidang perencanaan, dengan meningkatkan tingkat kinerja sistem perencanaan, dengan mengklarifikasi masalah representasional dan kombinatorial yang terlibat, dan dengan pengembangan heuristik yang berguna. Namun, ada pertanyaan tentang seberapa jauh teknik ini akan berkembang. Tampaknya kemajuan lebih lanjut pada masalah yang lebih besar tidak dapat hanya bergantung pada representasi yang difaktorkan dan proposisional, dan akan memerlukan semacam sintesis representasi orde pertama dan hierarkis dengan heuristik efisien yang saat ini digunakan.

1.6 RINGKASAN

Dalam bab ini, kami mendefinisikan masalah perencanaan dalam lingkungan statis yang deterministik, dapat diamati sepenuhnya. Kami menjelaskan representasi PDDL untuk masalah perencanaan dan beberapa pendekatan algoritmik untuk menyelesaikannya. Hal-hal yang perlu diingat:

- Sistem perencanaan adalah algoritme pemecahan masalah yang beroperasi pada representasi proposisional atau relasional eksplisit dari keadaan dan tindakan. Representasi ini memungkinkan derivasi heuristik yang efektif dan pengembangan algoritme yang kuat dan fleksibel untuk memecahkan masalah.
- PDDL, Bahasa Definisi Domain Perencanaan, menjelaskan keadaan awal dan tujuan sebagai konjungsi literal, dan tindakan dalam hal prasyarat dan efeknya.
- Pencarian ruang keadaan dapat beroperasi dalam arah maju (kemajuan) atau arah mundur (regresi). Heuristik yang efektif dapat diturunkan dengan asumsi independensi subtujuan dan dengan berbagai relaksasi masalah perencanaan.
- Grafik perencanaan dapat dibangun secara bertahap, dimulai dari keadaan awal. Setiap lapisan berisi superset dari semua literal atau tindakan yang dapat terjadi pada langkah waktu tersebut dan mengodekan hubungan pengecualian bersama (mutex) di antara literal atau tindakan yang tidak dapat terjadi bersamaan. Grafik perencanaan menghasilkan heuristik yang berguna untuk perencana ruang-keadaan dan orde parsial dan dapat digunakan secara langsung dalam algoritma GRAPHPLAN.
- Pendekatan lain mencakup deduksi orde pertama atas aksioma kalkulus situasi; mengodekan masalah perencanaan sebagai masalah pemuasan Boolean atau sebagai

masalah pemuasan kendala; dan secara eksplisit mencari melalui ruang rencana yang diurutkan sebagian.

- Setiap pendekatan utama untuk perencanaan memiliki penganutnya, dan belum ada konsensus tentang mana yang terbaik. Persaingan dan pemupukan silang di antara pendekatan telah menghasilkan peningkatan signifikan dalam efisiensi untuk sistem perencanaan.

BAB 2

PERENCANAAN DAN TINDAKAN DI DUNIA NYATA

2.1 WAKTU, JADWAL, DAN SUMBER DAYA

Representasi perencanaan klasik membahas tentang apa yang harus dilakukan, dan dalam urutan apa, tetapi representasi tersebut tidak dapat membahas tentang waktu: berapa lama suatu tindakan berlangsung dan kapan tindakan itu terjadi. Misalnya, para perencana Bab 10 dapat membuat jadwal untuk maskapai penerbangan yang menyebutkan pesawat mana yang ditugaskan untuk penerbangan mana, tetapi kita juga perlu mengetahui waktu keberangkatan dan kedatangan. Ini adalah pokok bahasan penjadwalan. Dunia nyata juga memberlakukan banyak kendala sumber daya; misalnya, maskapai penerbangan memiliki jumlah staf yang terbatas—dan staf yang berada di satu penerbangan tidak dapat berada di penerbangan lain pada waktu yang sama. Bagian ini membahas metode untuk merepresentasikan dan memecahkan masalah perencanaan yang mencakup kendala waktu dan sumber daya.

Pendekatan yang kami ambil dalam bagian ini adalah "rencanakan dulu, jadwalkan kemudian": yaitu, kami membagi keseluruhan masalah menjadi fase perencanaan di mana tindakan dipilih, dengan beberapa kendala urutan, untuk memenuhi tujuan masalah, dan fase penjadwalan selanjutnya, di mana informasi waktu ditambahkan ke rencana untuk memastikan bahwa rencana tersebut memenuhi kendala sumber daya dan tenggat waktu.

```
Jobs({AddEngine1 < AddWheels1 < Inspect1 },
      {AddEngine2 < AddWheels2 < Inspect2 })
```

```
Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))
```

```
Action(AddEngine1, DURATION: 30, USE: EngineHoists(1))
```

```
Action(AddEngine2, DURATION: 60, USE: EngineHoists(1))
```

```
Action(AddWheels1, DURATION: 30, CONSUME: LugNuts(20), USE: WheelStations(1))
```

```
Action(AddWheels2, DURATION: 15, CONSUME: LugNuts(20), USE: WheelStations(1))
```

```
Action(Inspect i, DURATION: 10, USE: Inspectors(1))
```

Gambar 2.1 Masalah penjadwalan job-shop untuk merakit dua mobil, dengan keterbatasan sumber daya. Notasi $A < B$ berarti bahwa tindakan A harus mendahului tindakan B .

Pendekatan ini umum dalam pengaturan logistik dan manufaktur dunia nyata, di mana fase perencanaan sering dilakukan oleh para ahli manusia. Metode otomatis Bab 10 juga dapat digunakan untuk fase perencanaan, asalkan metode tersebut menghasilkan rencana dengan batasan pemesanan minimal yang diperlukan untuk ketepatan. G RAPHPLAN, SATPLAN, dan perencana pesanan parsial dapat melakukan ini; metode berbasis pencarian menghasilkan rencana yang dipesan sepenuhnya, tetapi ini dapat dengan mudah diubah menjadi rencana dengan batasan pemesanan minimal.

Menggambarkan Kendala Temporal dan Sumber Daya

Permasalahan penjadwalan job-shop yang umum, terdiri dari serangkaian pekerjaan, yang masing-masing terdiri dari kumpulan tindakan dengan kendala pemesanan di antaranya. Setiap tindakan memiliki durasi dan serangkaian kendala sumber daya yang diperlukan oleh tindakan tersebut. Setiap kendala menentukan jenis sumber daya (misalnya, baut, kunci inggris, atau pilot), jumlah sumber daya yang diperlukan, dan apakah sumber daya tersebut dapat dikonsumsi (misalnya, baut tidak lagi tersedia untuk digunakan) atau dapat digunakan kembali (misalnya, seorang pilot sedang bertugas selama penerbangan tetapi tersedia lagi saat penerbangan selesai).

Sumber daya juga dapat diproduksi oleh tindakan dengan konsumsi negatif, termasuk tindakan manufaktur, pengembangan, dan pasokan ulang. Solusi untuk permasalahan penjadwalan job-shop harus menentukan waktu mulai untuk setiap tindakan dan harus memenuhi semua kendala pemesanan temporal dan kendala sumber daya. Seperti halnya masalah pencarian dan perencanaan, solusi dapat dievaluasi menurut fungsi biaya; ini bisa sangat rumit, dengan biaya sumber daya nonlinier, biaya penundaan tergantung waktu, dan sebagainya. Untuk menyederhanakannya, kami berasumsi bahwa fungsi biaya hanyalah durasi total rencana, yang disebut makespan.

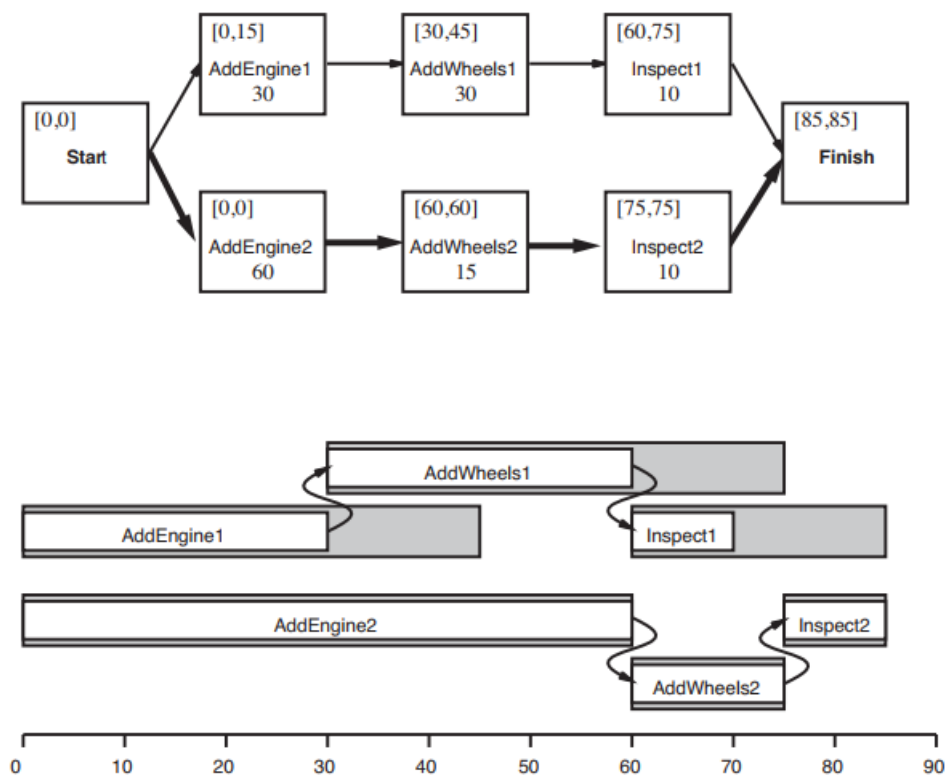
Gambar 2.1 menunjukkan contoh sederhana: masalah yang melibatkan perakitan dua mobil. Masalah tersebut terdiri dari dua pekerjaan, masing-masing dalam bentuk *[AddEngine, AddWheels, Inspect]*. Kemudian pernyataan *Resources* menyatakan bahwa ada empat jenis sumber daya, dan memberikan jumlah masing-masing jenis yang tersedia di awal: 1 kerekan mesin, 1 stasiun roda, 2 inspektur, dan 500 mur roda. Skema tindakan memberikan durasi dan kebutuhan sumber daya untuk setiap tindakan. Mur roda digunakan saat roda ditambahkan ke mobil, sedangkan sumber daya lainnya "dipinjam" di awal tindakan dan dilepaskan di akhir tindakan.

Representasi sumber daya sebagai kuantitas numerik, seperti *Inspectors(2)*, bukan sebagai entitas bernama, seperti *Inspector(I₁)* dan *Inspector(I₂)*, adalah contoh teknik yang sangat umum yang disebut agregasi. Ide utama agregasi adalah mengelompokkan objek individual ke dalam kuantitas ketika semua objek tidak dapat dibedakan sehubungan dengan tujuan yang ada. Dalam masalah perakitan kita, tidak masalah inspektur mana yang memeriksa mobil, jadi tidak perlu membuat perbedaan.

Agregasi sangat penting untuk mengurangi kompleksitas. Pertimbangkan apa yang terjadi ketika jadwal yang diusulkan memiliki 10 tindakan Inspeksi bersamaan tetapi hanya 9 inspektur yang tersedia. Dengan inspektur yang direpresentasikan sebagai kuantitas, kegagalan segera terdeteksi dan algoritme mundur untuk mencoba jadwal lain. Dengan inspektur yang direpresentasikan sebagai individu, algoritme mundur untuk mencoba semua 10! cara menugaskan inspektur ke tindakan.

Memecahkan Masalah Penjadwalan

Kita mulai dengan hanya mempertimbangkan masalah penjadwalan temporal, mengabaikan kendala sumber daya. Untuk meminimalkan makespan (durasi rencana), kita harus menemukan waktu mulai paling awal untuk semua tindakan yang konsisten dengan kendala pemesanan yang disertakan dengan masalah tersebut. Akan sangat membantu jika kendala pemesanan ini dilihat sebagai grafik berarah yang menghubungkan tindakan, seperti yang ditunjukkan pada Gambar 2.2.



Gambar 2.2 Atas: representasi kendala temporal untuk masalah penjadwalan job-shop pada Gambar 2.1. Durasi setiap tindakan diberikan di bagian bawah setiap persegi panjang. Dalam menyelesaikan masalah, kami menghitung waktu mulai paling awal dan paling akhir sebagai pasangan $[ES, LS]$, yang ditampilkan di kiri atas. Perbedaan antara kedua angka ini adalah kelonggaran suatu tindakan; tindakan dengan kelonggaran nol berada di jalur kritis, yang ditunjukkan dengan panah tebal. Bawah: solusi yang sama ditunjukkan sebagai garis waktu. Persegi panjang abu-abu mewakili interval waktu di mana suatu tindakan dapat dieksekusi,

asalkan kendala pemesanan dipatuhi. Bagian persegi panjang abu-abu yang tidak ditempati menunjukkan kelonggaran.

Kita dapat menerapkan metode jalur kritis (CPM) pada grafik ini untuk menentukan kemungkinan waktu mulai dan berakhirnya setiap tindakan. Jalur melalui grafik yang mewakili rencana pesanan parsial adalah urutan tindakan yang diurutkan secara linier yang dimulai dengan Mulai dan diakhiri dengan Selesai.

Jalur kritis adalah jalur yang durasi totalnya paling lama; jalur tersebut "kritis" karena menentukan durasi seluruh rencana—memperpendek jalur lain tidak memperpendek rencana secara keseluruhan, tetapi menunda dimulainya tindakan apa pun pada jalur kritis memperlambat seluruh rencana. Tindakan yang berada di luar jalur kritis memiliki rentang waktu untuk dapat dieksekusi. Rentang waktu tersebut ditentukan dalam bentuk waktu mulai paling awal, ES , dan waktu mulai paling akhir, LS . Besaran $LS - ES$ dikenal sebagai kelonggaran suatu tindakan. Kita dapat melihat pada Gambar 2.2 bahwa keseluruhan rencana akan memakan waktu 85 menit, bahwa setiap tindakan pada pekerjaan teratas memiliki kelonggaran selama 15 menit, dan bahwa setiap tindakan pada jalur kritis tidak memiliki kelonggaran (menurut definisi). Bersama-sama, waktu ES dan LS untuk semua tindakan tersebut merupakan jadwal untuk masalah tersebut.

Rumus berikut berfungsi sebagai definisi untuk ES dan LS dan juga sebagai garis besar algoritma pemrograman dinamis untuk menghitungnya. A dan B adalah tindakan, dan $A < B$ berarti bahwa A muncul sebelum B :

$$ES(Start) = 0$$

$$ES(B) = \max_{A < B} ES(A) + Duration(A)$$

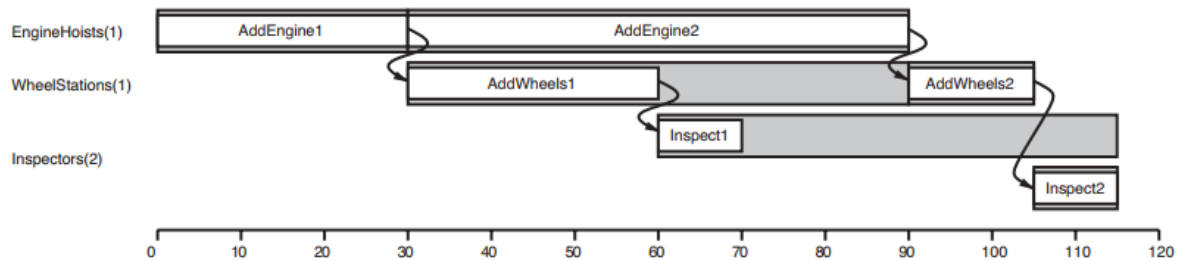
$$LS(Finish) = ES(Finish)$$

$$LS(A) = \max_{B < A} LS(B) - Duration(A).$$

Idenya adalah kita mulai dengan menetapkan $ES(Start)$ menjadi 0. Kemudian, segera setelah kita mendapatkan tindakan B sehingga semua tindakan yang datang tepat sebelum B memiliki nilai ES yang ditetapkan, kita tetapkan $ES(B)$ menjadi maksimum dari waktu penyelesaian paling awal dari tindakan-tindakan yang datang tepat sebelum itu, di mana waktu penyelesaian paling awal dari suatu tindakan didefinisikan sebagai waktu mulai paling awal ditambah durasinya. Proses ini berulang hingga setiap tindakan telah ditetapkan nilai ES . Nilai LS dihitung dengan cara yang sama, bekerja mundur dari tindakan $Finish$. Kompleksitas algoritma jalur kritis hanya $O(Nb)$, di mana N adalah jumlah tindakan dan b adalah faktor percabangan maksimum ke dalam atau keluar dari suatu tindakan. (Untuk melihat ini, perhatikan bahwa perhitungan LS dan ES dilakukan sekali untuk setiap tindakan, dan setiap perhitungan diulang paling banyak b tindakan lainnya.)

Oleh karena itu, menemukan jadwal durasi minimum, dengan urutan parsial pada tindakan dan tidak ada kendala sumber daya, cukup mudah. Secara matematis, masalah jalur kritis mudah dipecahkan karena didefinisikan sebagai gabungan pertidaksamaan linier pada

waktu mulai dan berakhir. Ketika kita memperkenalkan kendala sumber daya, kendala yang dihasilkan pada waktu mulai dan berakhir menjadi lebih rumit. Misalnya, tindakan *AddEngine*, yang dimulai pada waktu yang sama pada Gambar 2.2, memerlukan *EngineHoist* yang sama dan karenanya tidak dapat tumpang tindih. Kendala "tidak dapat tumpang tindih" adalah disjungsi dari dua pertidaksamaan linier, satu untuk setiap kemungkinan urutan. Pengenalan disjungsi ternyata membuat penjadwalan dengan kendala sumber daya menjadi NP-keras.



Gambar 2.3 Solusi untuk masalah penjadwalan job-shop dari Gambar 2.1, dengan mempertimbangkan keterbatasan sumber daya. Margin sebelah kiri mencantumkan tiga sumber daya yang dapat digunakan kembali, dan tindakan ditampilkan sejajar horizontal dengan sumber daya yang digunakan. Ada dua kemungkinan jadwal, tergantung pada perakitan mana yang menggunakan kerekan mesin terlebih dahulu; kami telah menunjukkan solusi dengan durasi terpendek, yang memerlukan waktu 115 menit.

Gambar 2.3 menunjukkan solusi dengan waktu penyelesaian tercepat, 115 menit. Ini 30 menit lebih lama dari 85 menit yang diperlukan untuk jadwal tanpa kendala sumber daya. Perhatikan bahwa tidak ada waktu di mana kedua inspektur diperlukan, jadi kita dapat segera memindahkan salah satu dari dua inspektur kita ke posisi yang lebih produktif. Kompleksitas penjadwalan dengan kendala sumber daya sering terlihat dalam praktik maupun teori. Masalah tantangan yang diajukan pada tahun 1963—untuk menemukan jadwal optimal untuk masalah yang melibatkan hanya 10 mesin dan 10 pekerjaan dengan 100 tindakan masing-masing—tidak terpecahkan selama 23 tahun.

Banyak pendekatan telah dicoba, termasuk cabang-dan-batas, simulasi anil, pencarian tabu, pemenuhan kendala, dan teknik lain. Satu heuristik sederhana tetapi populer adalah algoritma kelonggaran minimum: pada setiap iterasi, jadwalkan untuk permulaan paling awal yang memungkinkan tindakan tidak terjadwal mana pun yang memiliki semua pendahulunya yang dijadwalkan dan memiliki kelonggaran paling sedikit; kemudian perbarui waktu *ES* dan *LS* untuk setiap tindakan yang terpengaruh dan ulangi. Heuristik tersebut menyerupai heuristik *minimum-remaining-values* (MRV) dalam pemenuhan kendala. Seringkali berhasil dengan baik dalam praktik, tetapi untuk masalah perakitan kita menghasilkan solusi 130 menit, bukan solusi 115 menit seperti pada Gambar 2.3.

Sampai pada titik ini, kami berasumsi bahwa serangkaian tindakan dan batasan pemesanan sudah ditetapkan. Berdasarkan asumsi ini, setiap masalah penjadwalan dapat dipecahkan dengan urutan yang tidak tumpang tindih yang menghindari semua konflik sumber

daya, asalkan setiap tindakan dapat dilakukan sendiri. Namun, jika masalah penjadwalan terbukti sangat sulit, mungkin bukan ide yang baik untuk menyelesaikannya dengan cara ini—mungkin lebih baik untuk mempertimbangkan kembali tindakan dan batasan, jika hal itu mengarah pada masalah penjadwalan yang jauh lebih mudah.

Jadi, masuk akal untuk mengintegrasikan perencanaan dan penjadwalan dengan mempertimbangkan durasi dan tumpang tindih selama penyusunan rencana pesanan parsial. Beberapa algoritme perencanaan dapat ditambah untuk menangani informasi ini. Misalnya, rencana pesanan parsial dapat mendeteksi pelanggaran batasan sumber daya dengan cara yang sama seperti mereka mendeteksi konflik dengan hubungan sebab akibat. Heuristik dapat dirancang untuk memperkirakan total waktu penyelesaian suatu rencana. Ini saat ini merupakan bidang penelitian yang aktif.

2.2 PERENCANAAN HIRARKIS

Semua metode pemecahan masalah dan perencanaan pada bab-bab sebelumnya beroperasi dengan serangkaian tindakan atom yang tetap. Tindakan dapat dirangkai menjadi rangkaian atau jaringan percabangan; algoritme canggih dapat menghasilkan solusi yang berisi ribuan tindakan.

Untuk rencana yang dijalankan oleh otak manusia, tindakan atom adalah aktivasi otot. Dalam angka yang sangat bulat, kita memiliki sekitar 10^3 otot untuk diaktifkan (639, menurut beberapa hitungan, tetapi banyak di antaranya memiliki beberapa subunit); kita dapat memodulasi aktivasinya mungkin 10 kali per detik; dan kita hidup dan terjaga selama sekitar 10^9 detik secara keseluruhan.

Jadi, kehidupan manusia berisi sekitar 10^{13} tindakan, lebih atau kurang satu atau dua orde besaran. Bahkan jika kita membatasi diri untuk merencanakan dalam jangka waktu yang jauh lebih pendek—misalnya, liburan dua minggu di Hawaii—rencana motorik terperinci akan berisi sekitar 10^{10} tindakan. Jumlahnya jauh lebih banyak dari 1000.

Untuk menjembatani kesenjangan ini, sistem AI mungkin harus melakukan apa yang tampaknya dilakukan manusia: merencanakan pada tingkat abstraksi yang lebih tinggi. Rencana yang masuk akal untuk liburan Hawaii mungkin adalah "Pergi ke bandara San Francisco; naik penerbangan Hawaiian Airlines 11 ke Honolulu; lakukan kegiatan liburan selama dua minggu; naik penerbangan Hawaiian Airlines 12 kembali ke San Francisco; pulang." Dengan rencana seperti itu, tindakan "Pergi ke bandara San Francisco" dapat dilihat sebagai tugas perencanaan itu sendiri, dengan solusi seperti "Berkendara ke tempat parkir jangka panjang; parkir; naik shuttle ke terminal." Setiap tindakan ini, pada gilirannya, dapat diuraikan lebih lanjut, hingga kita mencapai tingkat tindakan yang dapat dieksekusi tanpa pertimbangan untuk menghasilkan urutan kontrol motorik yang diperlukan.

Dalam contoh ini, kita melihat bahwa perencanaan dapat terjadi sebelum dan selama eksekusi rencana; misalnya, seseorang mungkin akan menunda masalah perencanaan rute dari tempat parkir di tempat parkir jangka panjang ke halte bus shuttle hingga tempat parkir tertentu ditemukan selama eksekusi. Dengan demikian, tindakan khusus itu akan tetap berada pada level abstrak sebelum fase eksekusi.

Di sini, kami berkonsentrasi pada aspek dekomposisi hierarkis, sebuah ide yang meliputi hampir semua upaya untuk mengelola kompleksitas. Misalnya, perangkat lunak yang kompleks dibuat dari hierarki subrutin atau kelas objek; tentara beroperasi sebagai hierarki unit; pemerintah dan perusahaan memiliki hierarki departemen, anak perusahaan, dan kantor cabang.

Manfaat utama dari struktur hierarkis adalah bahwa, pada setiap level hierarki, tugas komputasi, misi militer, atau fungsi administratif direduksi menjadi sejumlah kecil aktivitas di level yang lebih rendah berikutnya, sehingga biaya komputasi untuk menemukan cara yang benar untuk mengatur aktivitas tersebut untuk masalah saat ini menjadi kecil. Metode nonhierarkis, di sisi lain, mereduksi tugas menjadi sejumlah besar tindakan individu; untuk masalah skala besar, ini sama sekali tidak praktis.

Tindakan Tingkat Tinggi

Formalisasi dasar yang kami adopsi untuk memahami dekomposisi hierarkis berasal dari area jaringan tugas hierarkis atau perencanaan HTN. Seperti dalam perencanaan klasik, kami mengasumsikan observabilitas dan determinisme penuh serta ketersediaan serangkaian tindakan, yang sekarang disebut tindakan primitif, dengan skema prakondisi—efek standar.

Konsep tambahan utama adalah tindakan tingkat tinggi atau HLA—misalnya, tindakan “Pergi ke bandara San Francisco” dalam contoh yang diberikan sebelumnya. Setiap HLA memiliki satu atau beberapa kemungkinan penyempurnaan, menjadi urutan tindakan, yang masing-masing dapat berupa HLA atau tindakan primitif (yang tidak memiliki penyempurnaan menurut definisi). Misalnya, tindakan “Pergi ke bandara San Francisco,” yang secara formal direpresentasikan sebagai $Go(Home, SFO)$, mungkin memiliki dua kemungkinan penyempurnaan, seperti yang ditunjukkan pada Gambar 2.4.

```
Refinement( $Go(Home, SFO)$ ,
  STEPS: [ $Drive(Home, SFO, LongTermParking)$ ,
     $Shuttle(SFO, LongTermParking, SFO)$ ] )
Refinement( $Go(Home, SFO)$ ,
  STEPS: [ $Taxi(Home, SFO)$ ] )

Refinement( $Navigate([a, b], [x, y])$ ,
  PRECOND:  $a = x \wedge b = y$ 
  STEPS: [ ] )
Refinement( $Navigate([a, b], [x, y])$ ,
  PRECOND:  $Connected([a, b], [a - 1, b])$ 
  STEPS: [ $Left, Navigate([a - 1, b], [x, y])$ ] )
Refinement( $Navigate([a, b], [x, y])$ ,
  PRECOND:  $Connected([a, b], [a + 1, b])$ 
  STEPS: [ $Right, Navigate([a + 1, b], [x, y])$ ] )
```

Gambar 2.4 Definisi penyempurnaan yang mungkin untuk dua tindakan tingkat tinggi: pergi ke bandara San Francisco dan bernavigasi di dunia hampa. Dalam kasus terakhir, perhatikan sifat rekursif dari penyempurnaan dan penggunaan prasyarat.

Gambar yang sama menunjukkan penyempurnaan rekursif untuk navigasi di dunia vakum: untuk mencapai tujuan, ambil langkah, lalu pergi ke tujuan. Contoh-contoh ini menunjukkan bahwa tindakan tingkat tinggi dan penyempurnaannya mewujudkan pengetahuan tentang cara melakukan sesuatu. Misalnya, penyempurnaan untuk $Go(Home, SFO)$ menyatakan bahwa untuk sampai ke bandara Anda dapat berkendara atau naik taksi; membeli susu, duduk, dan memindahkan kuda ke e4 tidak perlu dipertimbangkan.

Penyempurnaan HLA yang hanya berisi tindakan primitif disebut implementasi HLA. Misalnya, dalam dunia vakum, urutan $[Right, Right, Down]$ dan $[Down, Right, Right]$ keduanya mengimplementasikan HLA $Navigate([1, 3], [3, 2])$. Implementasi rencana tingkat tinggi (urutan HLA) adalah gabungan implementasi setiap HLA dalam urutan tersebut. Mengingat definisi prakondisi–efek dari setiap tindakan primitif, mudah untuk menentukan apakah implementasi rencana tingkat tinggi tertentu mencapai tujuan.

Kita dapat mengatakan, kemudian, bahwa rencana tingkat tinggi mencapai tujuan dari keadaan tertentu jika setidaknya satu implementasinya mencapai tujuan dari keadaan tersebut. "Setidaknya satu" dalam definisi ini sangat penting—tidak semua implementasi harus mencapai tujuan, karena agen dapat memutuskan implementasi mana yang akan dijalankannya. Dengan demikian, serangkaian implementasi yang mungkin dalam perencanaan HTN—yang masing-masing mungkin memiliki hasil yang berbeda—tidak sama dengan serangkaian hasil yang mungkin dalam perencanaan nondeterministik. Di sana, kami mensyaratkan agar suatu rencana berfungsi untuk semua hasil karena agen tidak dapat memilih hasilnya; alam yang menentukannya.

Kasus paling sederhana adalah HLA yang memiliki tepat satu implementasi. Dalam kasus tersebut, kita dapat menghitung prasyarat dan efek HLA dari implementasi tersebut dan kemudian memperlakukan HLA tersebut persis seolah-olah merupakan tindakan primitif itu sendiri. Dapat ditunjukkan bahwa kumpulan HLA yang tepat dapat menghasilkan kompleksitas waktu pencarian buta yang menurun dari eksponensial pada kedalaman solusi menjadi linear pada kedalaman solusi, meskipun merancang kumpulan HLA semacam itu sendiri mungkin merupakan tugas yang tidak mudah. Ketika HLA memiliki beberapa kemungkinan implementasi, ada dua pilihan: pertama adalah mencari di antara implementasi tersebut satu yang berfungsi, yang kedua adalah bernalar secara langsung tentang HLA—terlepas dari banyaknya implementasi. Metodeterakhir memungkinkan derivasi rencana abstrak yang terbukti benar, tanpa perlu mempertimbangkan implementasinya.

Mencari Solusi Primitif

Perencanaan HTN sering diformulasikan dengan satu tindakan "tingkat atas" yang disebut *Act*, yang tujuannya adalah untuk menemukan implementasi *Act* yang mencapai tujuan. Pendekatan ini sepenuhnya umum. Sebagai contoh, permasalahan perencanaan klasik dapat didefinisikan sebagai berikut: untuk setiap tindakan primitif a_i , berikan satu penyempurnaan *Act* dengan langkah-langkah $[a_i, Act]$. Hal ini menciptakan definisi rekursif *Act* yang memungkinkan kita menambahkan tindakan. Namun, kita memerlukan cara untuk menghentikan rekursi; kita melakukannya dengan menyediakan satu penyempurnaan lagi untuk *Act*, satu dengan daftar langkah kosong dan dengan prasyarat yang sama dengan tujuan

permasalahan. Ini berarti bahwa jika tujuan telah tercapai, maka implementasi yang tepat adalah tidak melakukan apa pun.

Pendekatan ini mengarah pada algoritma sederhana: pilih HLA berulang kali dalam rencana saat ini dan ganti dengan salah satu penyempurnaannya, hingga rencana tersebut mencapai tujuan. Salah satu kemungkinan implementasi berdasarkan pencarian pohon breadth-first ditunjukkan pada Gambar 2.5. Rencana dipertimbangkan berdasarkan kedalaman penumpukan penyempurnaan, alih-alih jumlah langkah primitif. Merancang versi pencarian grafik dari algoritma ini, serta versi pencarian mendalam dan pendalaman iteratif, sangatlah mudah. Intinya, bentuk pencarian hierarkis ini mengeksplorasi ruang urutan yang sesuai dengan pengetahuan yang terkandung dalam pustaka HLA tentang bagaimana segala sesuatunya harus dilakukan. Banyak pengetahuan yang dapat dikodekan, tidak hanya dalam urutan tindakan yang ditentukan dalam setiap penyempurnaan, tetapi juga dalam prasyarat penyempurnaan. Untuk beberapa domain, perencana HTN telah mampu menghasilkan rencana besar dengan pencarian yang sangat sedikit. Misalnya, O-PLAN, yang menggabungkan perencanaan HTN dengan penjadwalan, telah digunakan untuk mengembangkan rencana produksi untuk Hitachi. Masalah umum melibatkan lini produk yang terdiri dari 350 produk berbeda, 35 mesin perakitan, dan lebih dari 2000 operasi berbeda. Perencana tersebut menghasilkan jadwal 30 hari dengan tiga shift 8 jam sehari, yang melibatkan puluhan juta langkah. Aspek penting lainnya dari rencana HTN adalah bahwa rencana tersebut, menurut definisinya, terstruktur secara hierarkis; biasanya hal ini membuatnya mudah dipahami manusia.

function *HIERARCHICA – SEARCH*(*problem, hierarchy*) **returns** *a solution, or failure*

frontier $\leftarrow$ a FIFO queue with [*Act*] as the only element

loop do

if *EMPTY?*(*frontier*) **then return failure**

plan $\leftarrow$ *POP*(*frontier*) /* chooses the shallowest plan in *frontier* */

hla $\leftarrow$ the first HLA in *plan*, or *null* if none

prefix, suffix $\leftarrow$ the action subsequences before and after *hla* in *plan*

outcome $\leftarrow$ *RESULT*(*problem*.*INITIAL – STATE*, *prefix*)

if *hla* is *null* **then** /* so *plan* is primitive and *outcome* is its result */

if *outcome* satisfies *problem*.*GOAL* **then return plan**

else for each *sequence* in *REFINEMENTS*(*hla, outcome, hierarchy*) **do**

frontier $\leftarrow$ *INSERT*(*APPEND*(*prefix, sequence, suffix*), *frontier*)

Gambar 2.5 Implementasi menyeluruh dari pencarian perencanaan ke depan hierarkis.

Rencana awal yang diberikan ke algoritme adalah [*Act*]. Fungsi *REFINEMENTS* mengembalikan serangkaian urutan tindakan, satu untuk setiap penyempurnaan HLA yang prasyaratnya dipenuhi oleh status, hasil yang ditentukan.

Manfaat komputasional dari pencarian hierarkis dapat dilihat dengan memeriksa kasus yang diidealkan. Misalkan masalah perencanaan memiliki solusi dengan *d* tindakan primitif. Untuk

perencana ruang keadaan maju nonhierarkis dengan b tindakan yang diizinkan di setiap keadaan, biayanya adalah $O(b^d)$. Untuk perencana HTN, mari kita asumsikan struktur penyempurnaan yang sangat teratur: setiap tindakan nonprimitif memiliki r penyempurnaan yang mungkin, masing-masing menjadi k tindakan pada tingkat yang lebih rendah berikutnya. Kita ingin mengetahui berapa banyak pohon penyempurnaan yang berbeda dengan struktur ini.

Sekarang, jika ada d tindakan pada tingkat primitif, maka jumlah tingkat di bawah akar adalah $\log_k d$, jadi jumlah simpul penyempurnaan internal adalah $1 + k + k^2 + \dots + k^{\log_k d - 1} = (d - 1)/(k - 1)$. Setiap simpul internal memiliki r penyempurnaan yang mungkin, jadi $r^{(d-1)/(k-1)}$ pohon dekomposisi reguler yang mungkin dapat dibangun. Dengan memeriksa rumus ini, kita melihat bahwa menjaga r tetap kecil dan k tetap besar dapat menghasilkan penghematan yang sangat besar: pada dasarnya kita mengambil akar ke- k dari biaya nonhierarkis, jika b dan r sebanding. r kecil dan k besar berarti pustaka HLA dengan sejumlah kecil penyempurnaan yang masing-masing menghasilkan urutan tindakan yang panjang (yang tetap memungkinkan kita untuk memecahkan masalah apa pun). Ini tidak selalu memungkinkan: urutan tindakan panjang yang dapat digunakan di berbagai masalah sangatlah berharga.

Kunci perencanaan HTN adalah pembangunan pustaka rencana yang berisi metode yang diketahui untuk mengimplementasikan tindakan tingkat tinggi yang kompleks. Salah satu metode untuk membangun pustaka adalah mempelajari metode tersebut dari pengalaman pemecahan masalah. Setelah pengalaman yang melelahkan dalam membangun rencana dari awal, agen dapat menyimpan rencana tersebut di pustaka sebagai metode untuk mengimplementasikan tindakan tingkat tinggi yang ditentukan oleh tugas. Dengan cara ini, agen dapat menjadi semakin kompeten dari waktu ke waktu karena metode baru dibangun di atas metode lama. Salah satu aspek penting dari proses pembelajaran ini adalah kemampuan untuk menggeneralisasi metode yang dibangun, menghilangkan detail yang khusus untuk contoh masalah (misalnya, nama pembangun atau alamat sebidang tanah) dan hanya mempertahankan elemen-elemen utama dari rencana. Bagi kami, tampaknya tidak masuk akal bahwa manusia dapat menjadi sekompeten mereka tanpa mekanisme semacam itu.

Mencari Solusi Abstrak

Algoritma pencarian hierarkis di bagian sebelumnya menyempurnakan HLA hingga ke urutan tindakan primitif untuk menentukan apakah suatu rencana dapat dilaksanakan. Hal ini bertentangan dengan akal sehat: seseorang seharusnya dapat menentukan bahwa rencana tingkat tinggi dua-HLA

[Drive(Home, SFOLongTermParking), Shuttle(SFOLongTermParking, SFO)]

membawa seseorang ke bandara tanpa harus menentukan rute yang tepat, pilihan tempat parkir, dan sebagainya. Solusinya tampak jelas: tulis deskripsi prakondisi-efek dari HLA, sama seperti kita menuliskan apa yang dilakukan tindakan primitif. Dari deskripsi tersebut, seharusnya mudah untuk membuktikan bahwa rencana tingkat tinggi mencapai sasaran. Ini

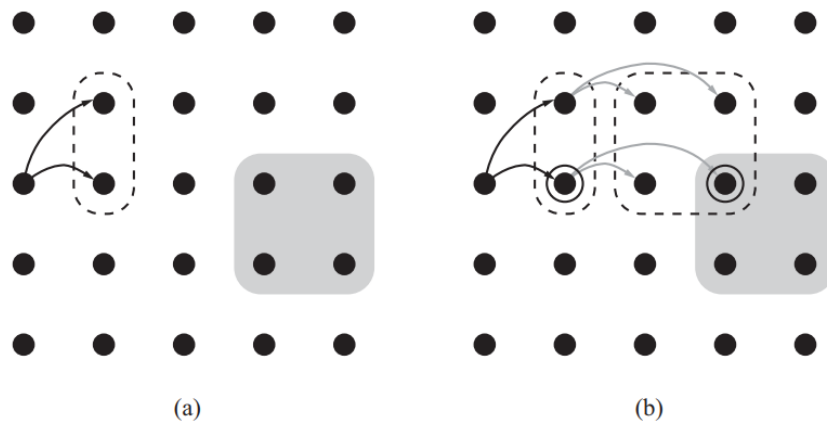
adalah tujuan utama perencanaan hierarkis, karena jika kita memperoleh rencana tingkat tinggi yang terbukti mencapai sasaran, bekerja dalam ruang pencarian kecil dari tindakan tingkat tinggi, maka kita dapat berkomitmen pada rencana itu dan mengerjakan masalah penyempurnaan setiap langkah rencana. Ini memberi kita pengurangan eksponensial yang kita cari.

Agar ini berhasil, harus terjadi bahwa setiap rencana tingkat tinggi yang "mengklaim" mencapai sasaran (berdasarkan deskripsi langkah-langkahnya) benar-benar mencapai sasaran dalam pengertian yang dijelaskan sebelumnya: harus memiliki setidaknya satu implementasi yang benar-benar mencapai sasaran. Properti ini disebut properti penyempurnaan ke bawah untuk deskripsi HLA.

Menulis deskripsi HLA yang memenuhi sifat penyempurnaan ke bawah, pada prinsipnya, mudah: selama deskripsi tersebut benar, maka setiap rencana tingkat tinggi yang mengklaim mencapai tujuan tersebut harus benar-benar melakukannya—jika tidak, deskripsi tersebut membuat klaim palsu tentang apa yang dilakukan HLA. Kita telah melihat cara menulis deskripsi yang benar untuk HLA yang memiliki tepat satu implementasi; masalah muncul ketika HLA memiliki beberapa implementasi. Bagaimana kita dapat menggambarkan efek dari suatu tindakan yang dapat diimplementasikan dengan berbagai cara? Satu jawaban yang aman (setidaknya untuk masalah di mana semua prasyarat dan tujuan bersifat positif) adalah dengan hanya menyertakan efek positif yang dicapai oleh setiap implementasi HLA dan efek negatif dari setiap implementasi.

Maka sifat penyempurnaan ke bawah akan terpenuhi. Sayangnya, semantik untuk HLA ini terlalu konservatif. Pertimbangkan lagi HLA $Go(Home, SFO)$, yang memiliki dua penyempurnaan, dan anggaplah, demi argumen, dunia sederhana di mana seseorang selalu dapat berkendara ke bandara dan parkir, tetapi naik taxi memerlukan Uang Tunai sebagai prasyarat. Dalam kasus tersebut, $Go(Home, SFO)$ tidak selalu membawa Anda ke bandara. Secara khusus, gagal jika Uang Tunai salah, jadi kita tidak dapat menyatakan $At(Agen, SFO)$ sebagai efek dari HLA. Namun, ini tidak masuk akal; jika agen tidak memiliki Uang Tunai, ia akan mengemudi sendiri. Mensyaratkan bahwa efek berlaku untuk setiap implementasi setara dengan mengasumsikan bahwa orang lain—musuh—akan memilih implementasi tersebut. Ini memperlakukan beberapa hasil HLA persis seperti jika HLA adalah tindakan nondeterministik. Untuk kasus kita, agen itu sendiri akan memilih implementasinya.

Komunitas bahasa pemrograman telah menciptakan istilah nondeterminisme demonik untuk kasus di mana musuh membuat pilihan, yang kontras dengan nondeterminisme malaikat, di mana agen itu sendiri yang membuat pilihan. Kami meminjam istilah ini untuk mendefinisikan semantik malaikat untuk deskripsi HLA. Konsep dasar yang diperlukan untuk memahami semantik malaikat adalah himpunan yang dapat dijangkau dari sebuah HLA: diberikan status s , himpunan yang dapat dijangkau untuk sebuah HLA h , ditulis sebagai $REACH(s, h)$, adalah himpunan status yang dapat dijangkau oleh salah satu implementasi HLA.



Gambar 2.6 Contoh skematis dari himpunan yang dapat dicapai. Himpunan status tujuan diarsir. Panah hitam dan abu-abu menunjukkan kemungkinan penerapan h_1 dan h_2 , masing-masing. (a) Himpunan yang dapat dicapai dari HLA h_1 dalam status s . (b) Himpunan yang dapat dicapai untuk urutan $[h_1, h_2]$. Karena ini berpotongan dengan himpunan tujuan, urutan tersebut mencapai tujuan.

Ide utamanya adalah bahwa agen dapat memilih elemen mana dari himpunan yang dapat dijangkau yang akan dimasukinya saat menjalankan HLA; dengan demikian, sebuah HLA dengan beberapa penyempurnaan lebih "kuat" daripada HLA yang sama dengan penyempurnaan yang lebih sedikit. Kita juga dapat mendefinisikan himpunan yang dapat dijangkau dari sekuens HLA. Misalnya, himpunan yang dapat dicapai dari suatu barisan $[h_1, h_2]$ adalah gabungan dari semua himpunan yang dapat dicapai yang diperoleh dengan menerapkan h_2 pada setiap keadaan dalam himpunan yang dapat dicapai dari h_1 :

$$\text{REACH}(s, [h_1, h_2]) = \bigcup_{s' \in \text{REACH}(s, h_1)} \text{REACH}(s', h_2).$$

Dengan definisi ini, rencana tingkat tinggi—urutan HLA—mencapai tujuan jika himpunan yang dapat dicapainya berpotongan dengan himpunan status tujuan. (Bandingkan ini dengan kondisi yang jauh lebih kuat untuk semantik demonik, di mana setiap anggota himpunan yang dapat dicapai harus menjadi status tujuan.) Sebaliknya, jika himpunan yang dapat dicapai tidak berpotongan dengan tujuan, maka rencana tersebut pasti tidak berhasil. Gambar 2.6 mengilustrasikan ide-ide ini.

Gagasan himpunan yang dapat dicapai menghasilkan algoritme yang mudah dipahami: mencari di antara rencana tingkat tinggi, mencari satu yang himpunan yang dapat dicapainya berpotongan dengan tujuan; setelah itu terjadi, algoritme dapat berkomitmen pada rencana abstrak itu, mengetahui bahwa rencana itu berhasil, dan fokus pada penyempurnaan rencana lebih lanjut. Kita akan kembali ke masalah algoritme nanti; pertama, kita pertimbangkan pertanyaan tentang bagaimana efek HLA—himpunan yang dapat dicapai untuk setiap kemungkinan status awal—diwakili. Seperti skema tindakan klasik, kita mewakili perubahan

yang dibuat pada setiap fluent. Anggap fluent sebagai variabel status. Tindakan primitif dapat menambahkan atau menghapus variabel atau membiarkannya tidak berubah.

HLA di bawah semantik malaikat dapat melakukan lebih banyak hal: ia dapat mengendalikan nilai variabel, mengaturnya menjadi benar atau salah tergantung pada implementasi mana yang dipilih. Faktanya, HLA dapat memiliki sembilan efek berbeda pada variabel: jika variabel dimulai dengan benar, ia dapat selalu membuatnya benar, selalu membuatnya salah, atau memiliki pilihan; jika variabel dimulai dengan salah, ia dapat selalu membuatnya salah, selalu membuatnya benar, atau memiliki pilihan; dan tiga pilihan untuk setiap kasus dapat digabungkan secara acak, sehingga menjadi sembilan. Secara notasi, ini sedikit menantang. Kita akan menggunakan simbol untuk berarti "mungkin, jika agen memilihnya." Jadi, efek $\tilde{\vdash} A$ berarti "mungkin menambahkan A ," yaitu, membiarkan A tidak berubah atau membuatnya benar. Demikian pula, $\simeq A$ berarti "mungkin menghapus A " dan $\tilde{\pm} A$ berarti "mungkin menambahkan atau menghapus A ."

Misalnya, HLA $Go(Home, SFO)$, dengan dua penyempurnaan yang ditunjukkan pada Gambar 2.4, mungkin menghapus Cash (jika agen memutuskan untuk naik taksi), sehingga seharusnya memiliki efek $\simeq Cash$. Dengan demikian, kita melihat bahwa deskripsi HLA dapat diturunkan, pada prinsipnya, dari deskripsi penyempurnaannya—sebenarnya, ini diperlukan jika kita menginginkan deskripsi HLA yang sebenarnya, sehingga sifat penyempurnaan menurun berlaku. Sekarang, misalkan kita memiliki skema berikut untuk HLA h_1 dan h_2 :

$$\begin{aligned} &Action(h_1, PRECOND: \neg A, EFFECT: A \wedge \simeq B), \\ &Action(h_2, PRECOND: \neg B, EFFECT: \tilde{\vdash} A \wedge \tilde{\pm} C). \end{aligned}$$

Artinya, h_1 menambahkan A dan mungkin menghapus B , sementara h_2 mungkin menambahkan A dan memiliki kendali penuh atas C . Sekarang, jika hanya B yang benar dalam keadaan awal dan tujuannya adalah $A \wedge C$ maka urutan $[h_1, h_2]$ mencapai tujuan: kita memilih implementasi h_1 yang membuat B salah, lalu memilih implementasi h_2 yang membiarkan A benar dan membuat C benar.

Pembahasan sebelumnya mengasumsikan bahwa efek HLA—himpunan yang dapat dicapai untuk setiap keadaan awal yang diberikan—dapat dijelaskan secara tepat dengan menjelaskan efek pada setiap variabel. Akan lebih baik jika ini selalu benar, tetapi dalam banyak kasus kita hanya dapat memperkirakan efeknya karena HLA mungkin memiliki implementasi yang tak terhingga banyaknya dan dapat menghasilkan himpunan yang dapat dicapai secara acak.

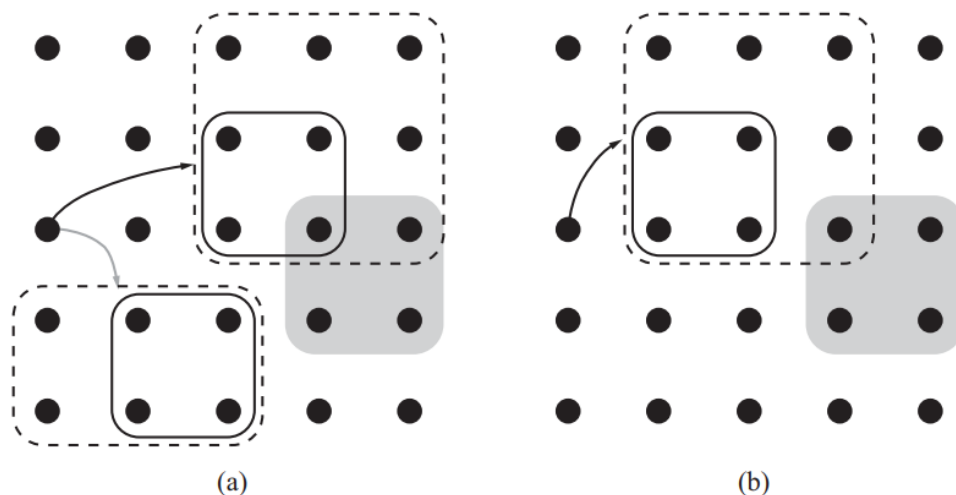
Misalnya, kita mengatakan bahwa $Go(Home, SFO)$ mungkin menghapus $Cash$; ia juga mungkin menambahkan $At(Car, SFO LongTermParking)$; tetapi ia tidak dapat melakukan keduanya—sebenarnya, ia harus melakukan satu hal. Seperti halnya dengan status keyakinan, kita mungkin perlu menulis deskripsi perkiraan. Kita akan menggunakan dua jenis perkiraan: deskripsi optimis $REACH^+(s, h)$ dari HLA h dapat melebihi-lebihkan himpunan yang dapat dicapai, sementara deskripsi pesimis $REACH^-(s, h)$ dapat mengecilkan himpunan yang dapat dicapai. Jadi, kita memiliki

$$\text{REACH}^-(s, h) \subseteq \text{REACH}(s, h) \subseteq \text{REACH}^+(s, h).$$

Misalnya, deskripsi optimis dari *Go(Home, SFO)* mengatakan bahwa ia mungkin menghapus *Cash* dan mungkin menambahkan *At(Car, SFO LongTermParking)*. Contoh bagus lainnya muncul dalam teka-teki 8, yang setengah dari statusnya tidak dapat dicapai dari status apa pun, deskripsi optimis dari *Act* mungkin mencakup seluruh ruang status, karena himpunan yang dapat dicapai secara tepat cukup goyah.

Dengan deskripsi perkiraan, pengujian untuk mengetahui apakah suatu rencana mencapai sasaran perlu dimodifikasi sedikit. Jika himpunan yang dapat dicapai secara optimis untuk rencana tersebut tidak berpotongan dengan sasaran, maka rencana tersebut tidak berhasil; jika himpunan yang dapat dicapai secara pesimis berpotongan dengan sasaran, maka rencana tersebut berhasil (Gambar 2.7(a)). Dengan deskripsi yang tepat, sebuah rencana bisa berhasil atau tidak, tetapi dengan deskripsi perkiraan, ada jalan tengah: jika himpunan optimis berpotongan dengan tujuan tetapi himpunan pesimis tidak, maka kita tidak dapat mengetahui apakah rencana tersebut berhasil (Gambar 2.7(b)).

Ketika keadaan ini muncul, ketidakpastian dapat diatasi dengan menyempurnakan rencana. Ini adalah situasi yang sangat umum dalam penalaran manusia. Misalnya, dalam merencanakan liburan Hawaii dua minggu yang disebutkan di atas, seseorang mungkin mengusulkan untuk menghabiskan dua hari di masing-masing dari tujuh pulau. Kehati-hatian akan menunjukkan bahwa rencana ambisius ini perlu disempurnakan dengan menambahkan rincian transportasi antarpulau.



Gambar 2.7 Pencapaian tujuan untuk rencana tingkat tinggi dengan deskripsi perkiraan. Rangkaian status tujuan diarsir. Untuk setiap rencana, rangkaian yang dapat dicapai pesimis (garis utuh) dan optimis (garis putus-putus) ditunjukkan. (a) Rencana yang ditunjukkan oleh panah hitam jelas mencapai tujuan, sedangkan rencana yang ditunjukkan oleh panah abu-abu jelas tidak. (b) Rencana yang perlu disempurnakan lebih lanjut untuk menentukan apakah rencana tersebut benar-benar mencapai tujuan.

Algoritma untuk perencanaan hierarkis dengan deskripsi malaikat perkiraan ditunjukkan pada Gambar 2.8. Untuk kesederhanaan, kami telah mempertahankan skema keseluruhan yang sama yang digunakan sebelumnya pada Gambar 2.5, yaitu, pencarian menyeluruh pertama dalam ruang penyempurnaan. Seperti yang baru saja dijelaskan, algoritme dapat mendeteksi rencana yang akan dan tidak akan berhasil dengan memeriksa perpotongan himpunan optimis dan pesimis yang dapat dicapai dengan tujuan.

Ketika rencana abstrak yang dapat dilaksanakan ditemukan, algoritme menguraikan masalah asli menjadi submasalah, satu untuk setiap langkah rencana. Keadaan awal dan tujuan untuk setiap submasalah diperoleh dengan meregresikan keadaan tujuan yang dijamin dapat dicapai melalui skema tindakan untuk setiap langkah rencana. Gambar 2.6(b) mengilustrasikan ide dasarnya: keadaan yang dilingkari di sebelah kanan adalah keadaan tujuan yang dijamin dapat dicapai, dan keadaan yang dilingkari di sebelah kiri adalah tujuan antara yang diperoleh dengan meregresikan tujuan melalui tindakan akhir.

```

function ANGELIC – SEARCH(problem, hierarchy, initialPlan) returns solution or fail

frontier  $\leftarrow$  a FIFO queue with initialPlan as the only element
loop do
  if EMPTY? (frontier) then return fail
  plan  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
  if REACH+(problem.INITIAL – STATE, plan) intersects problem.GOAL then
    if plan is primitive then return plan /* REACH+ is exact for primitive plans */
    guaranteed  $\leftarrow$  REACH– – (problem.INITIAL – STATE, plan)  $\cap$  problem.GOAL
    if guaranteed  $\neq \{ \}$  and MAKING – PROGRESS(plan, initialPlan) then
      finalState  $\leftarrow$  any element of guaranteed
      return DECOMPOSE(hierarchy, problem.INITIAL – STATE, plan, finalState)
    hla  $\leftarrow$  some HLA in plan
    prefix, suffix  $\leftarrow$  the action subsequences before and after hla in plan
    for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
      frontier  $\leftarrow$  INSERT(APPEND(prefix, sequence, suffix), frontier)



---


function DECOMPOSE(hierarchy, s0, plan, sf) returns a solution
solution  $\leftarrow$  an empty plan
while plan is not empty do
  action  $\leftarrow$  REMOVE – LAST(plan)
  si  $\leftarrow$  a state in REACH–(s0, plan) such that sf  $\in$  REACH–(si, action)
  problem  $\leftarrow$  a problem with INITIAL – STATE = si and GOAL = sf
  solution  $\leftarrow$  APPEND(ANGELIC – SEARCH(problem, hierarchy, action), solution)
  sf  $\leftarrow$  si
return solution

```

Gambar 2.8 Algoritma perencanaan hierarkis yang menggunakan semantik malaikat untuk mengidentifikasi dan berkomitmen pada rencana tingkat tinggi yang berhasil sambil menghindari rencana tingkat tinggi yang tidak berhasil. Predikat MAKING-PROGRESS memeriksa untuk memastikan bahwa kita tidak terjebak dalam regresi penyempurnaan yang tak terbatas. Di tingkat atas, panggil ANGELIC-SEARCH dengan [*Act*] sebagai *initialPlan*.

Kemampuan untuk berkomitmen atau menolak rencana tingkat tinggi dapat memberikan ANGELIC-SEARCH keuntungan komputasi yang signifikan atas HIERARCHICAL-SEARCH, yang pada gilirannya mungkin memiliki keuntungan besar atas BREADTH-FIRST-SEARCH biasa. Pertimbangkan, misalnya, membersihkan dunia vakum besar yang terdiri dari kamar-kamar persegi panjang yang dihubungkan oleh koridor sempit. Masuk akal untuk memiliki HLA untuk *Navigate* dan satu untuk *CleanWholeRoom*. (Membersihkan ruangan dapat diimplementasikan dengan penerapan berulang HLA lain untuk membersihkan setiap baris.)

Karena ada lima tindakan dalam domain ini, biaya untuk BREADTH-FIRST-SEARCH tumbuh sebagai 5^d , di mana d adalah panjang solusi terpendek (kira-kira dua kali jumlah total kotak); algoritme tidak dapat mengelola bahkan dua kamar 2×2 . PENCARIAN-HIERARKI lebih efisien, tetapi masih mengalami pertumbuhan eksponensial karena mencoba semua cara pembersihan yang konsisten dengan hierarki. PENCARIAN-ANGELIK berskala kira-kira linier dalam jumlah kotak—ia berkomitmen pada urutan tingkat tinggi yang baik dan memangkas opsi lainnya. Perhatikan bahwa membersihkan sekumpulan kamar dengan membersihkan setiap kamar secara bergiliran bukanlah ilmu roket: hal itu mudah bagi manusia justru karena struktur hierarki tugas tersebut.

Ketika kita mempertimbangkan betapa sulitnya manusia memecahkan teka-teki kecil seperti teka-teki 8, tampaknya kapasitas manusia untuk memecahkan masalah yang kompleks sebagian besar berasal dari keterampilan mereka dalam mengabstraksi dan menguraikan masalah untuk menghilangkan kombinatorik.

Pendekatan malaikat dapat diperluas untuk menemukan solusi berbiaya paling rendah dengan menggeneralisasi gagasan tentang himpunan yang dapat dicapai. Alih-alih keadaan dapat dicapai atau tidak, ia memiliki biaya untuk cara yang paling efisien untuk mencapainya. (Biaya adalah ∞ untuk status yang tidak dapat dijangkau.) Deskripsi optimis dan pesimis membatasi biaya ini. Dengan cara ini, pencarian malaikat dapat menemukan rencana abstrak yang terbukti optimal tanpa mempertimbangkan implementasinya.

Pendekatan yang sama dapat digunakan untuk memperoleh algoritme lookahead hierarkis yang efektif untuk pencarian daring, dalam gaya LRTA*. Dalam beberapa hal, algoritme tersebut mencerminkan aspek pertimbangan manusia dalam tugas-tugas seperti merencanakan liburan ke Hawaii—pertimbangan alternatif dilakukan pada awalnya pada tingkat abstrak dalam skala waktu yang panjang; beberapa bagian dari rencana dibiarkan cukup abstrak hingga waktu pelaksanaan, seperti cara menghabiskan dua hari malas di Molokai, sementara bagian lain direncanakan secara terperinci, seperti penerbangan yang akan diambil dan penginapan yang akan dipesan—tanpa penyempurnaan ini, tidak ada jaminan bahwa rencana tersebut akan layak.

2.3 PERENCANAAN DAN TINDAKAN DI WILAYAH NONDETERMINISTIK

Di bagian ini, kami memperluas perencanaan untuk menangani lingkungan yang sebagian dapat diamati, nondeterministik, dan tidak diketahui. Perencanaan tanpa sensor (juga dikenal sebagai perencanaan yang sesuai) untuk lingkungan tanpa pengamatan;

perencanaan kontinjensi untuk lingkungan yang sebagian dapat diamati dan nondeterministik; dan perencanaan dan perencanaan ulang daring untuk lingkungan yang tidak diketahui.

Meskipun konsep dasarnya sama, ada juga perbedaan yang signifikan. Ini muncul karena rencana berurusan dengan representasi yang difaktorkan daripada representasi atomik. Ini memengaruhi cara kami merepresentasikan kemampuan agen untuk bertindak dan melakukan pengamatan dan cara kami merepresentasikan status keyakinan—himpunan kemungkinan status fisik yang mungkin dialami agen—untuk lingkungan yang tidak dapat diamati dan sebagian dapat diamati. Kita juga dapat memanfaatkan banyak metode domain-independen untuk menghitung heuristik pencarian.

Pertimbangkan masalah ini: jika diberikan kursi dan meja, tujuannya adalah agar keduanya cocok—memiliki warna yang sama. Pada kondisi awal, kita memiliki dua kaleng cat, tetapi warna cat dan furniturnya tidak diketahui. Hanya meja yang awalnya berada dalam bidang pandang agen:

$$\begin{aligned} &Init(Object(Table) \wedge Object(Chair) \wedge Can(C1) \wedge Can(C2) \wedge InView(Table)) \\ &Goal(Color(Chair, c) \wedge Color(Table, c)) \end{aligned}$$

Ada dua tindakan: melepas tutup kaleng cat dan mengecat objek menggunakan cat dari kaleng yang terbuka. Skema tindakannya mudah, dengan satu pengecualian: kita sekarang mengizinkan prasyarat dan efek untuk memuat variabel yang bukan bagian dari daftar variabel tindakan. Artinya, $Paint(x, can)$ tidak menyebutkan variabel c , yang mewakili warna cat dalam kaleng. Dalam kasus yang dapat diamati sepenuhnya, hal ini tidak diperbolehkan—kita harus menamai tindakan itu $Paint(x, can, c)$. Namun dalam kasus yang dapat diamati sebagian, kita mungkin tahu atau tidak tahu warna apa yang ada di kaleng itu. (Variabel c dikuantifikasi secara universal, sama seperti semua variabel lain dalam skema tindakan.)

$$\begin{aligned} &Action(RemoveLid(can), \\ &\quad PRECOND: Can(can) \\ &\quad EFFECT: Open(can)) \\ &Action(Paint(x, can), \\ &\quad PRECOND: Object(x) \wedge Can(can) \wedge Color(can, c) \wedge Open(can) \\ &\quad EFFECT: Color(x, c)) \end{aligned}$$

Untuk memecahkan masalah yang dapat diamati sebagian, agen harus bernalar tentang persepsi yang akan diperolehnya saat menjalankan rencana. Persepsi akan diberikan oleh sensor agen saat benar-benar bertindak, tetapi saat merencanakan, agen akan memerlukan model sensornya. Model ini diberikan oleh suatu fungsi, $PERCEPT(s)$. Untuk perencanaan, kami melengkapi PDDL dengan jenis skema baru, skema persepsi:

$$\begin{aligned} &Percept(Color(x, c), \\ &\quad PRECOND: Object(x) \wedge InView(x)) \end{aligned}$$

Percept(*Color*(*can*, *c*),
 PRECOND: *Can*(*can*) $\wedge$ *InView*(*can*) $\wedge$ *Open*(*can*)

Skema pertama menyatakan bahwa setiap kali sebuah objek terlihat, agen akan melihat warna objek tersebut (yaitu, untuk objek x , agen akan mempelajari nilai kebenaran $Color(x, c)$ untuk semua c). Skema kedua menyatakan bahwa jika sebuah kaleng terbuka terlihat, maka agen akan melihat warna cat dalam kaleng tersebut. Karena tidak ada kejadian eksogen di dunia ini, warna sebuah objek akan tetap sama, meskipun tidak terlihat, hingga agen melakukan tindakan untuk mengubah warna objek tersebut. Tentu saja, agen akan memerlukan tindakan yang menyebabkan objek (satu per satu) terlihat:

Action(*LookAt*(x),
 PRECOND: *InView*(y) $\wedge$ ($x = y$)
 EFFECT: *InView*(x) $\wedge$ $\neg$ *InView*(y))

Untuk lingkungan yang dapat diamati sepenuhnya, kita akan memiliki aksioma *Percept* tanpa prasyarat untuk setiap fasih. Di sisi lain, agen tanpa sensor sama sekali tidak memiliki aksioma *Percept*. Perhatikan bahwa bahkan agen tanpa sensor dapat memecahkan masalah pengecatan. Salah satu solusinya adalah membuka kaleng cat apa pun dan mengoleskannya ke kursi dan meja, sehingga memaksa keduanya menjadi warna yang sama (meskipun agen tidak tahu apa warnanya).

Agen perencanaan kontingensi dengan sensor dapat menghasilkan rencana yang lebih baik. Pertama, lihat meja dan kursi untuk mengetahui warnanya; jika sudah sama, maka rencananya sudah selesai. Jika tidak, lihat kaleng cat; jika cat dalam kaleng memiliki warna yang sama dengan satu perabot, maka oleskan cat itu ke perabot lainnya. Jika tidak, cat kedua perabot dengan warna apa pun.

Terakhir, agen perencanaan daring mungkin menghasilkan rencana kontingensi dengan lebih sedikit cabang pada awalnya—mungkin mengabaikan kemungkinan bahwa tidak ada kaleng yang cocok dengan perabot mana pun—dan menangani masalah saat muncul dengan perencanaan ulang. Ia juga dapat menangani ketidaktepatan skema tindakannya. Sementara perencanaan kontingensi hanya berasumsi bahwa efek dari suatu tindakan selalu berhasil—bahwa mengecat kursi akan menyelesaikan tugasnya—agen perencanaan ulang akan memeriksa hasilnya dan membuat rencana tambahan untuk memperbaiki kegagalan yang tidak terduga, seperti area yang tidak dicat atau warna asli yang terlihat.

Di dunia nyata, agen menggunakan kombinasi pendekatan. Produsen mobil menjual ban serep dan kantung udara, yang merupakan perwujudan fisik dari cabang rencana kontingensi yang dirancang untuk menangani tusukan atau tabrakan. Di sisi lain, sebagian besar pengemudi mobil tidak pernah mempertimbangkan kemungkinan ini; ketika masalah muncul, mereka merespons sebagai agen perencanaan ulang.

Secara umum, agen hanya merencanakan kontingensi yang memiliki konsekuensi penting dan peluang yang tidak dapat diabaikan untuk terjadi. Jadi, pengemudi mobil yang mempertimbangkan perjalanan melintasi gurun Sahara harus membuat rencana kontingensi yang eksplisit untuk kerusakan, sedangkan perjalanan ke supermarket memerlukan lebih sedikit perencanaan di muka. Selanjutnya, kita akan melihat masing-masing dari ketiga pendekatan tersebut secara lebih rinci.

Perencanaan Tanpa Sensor

Ide dasar pencarian dalam ruang status keyakinan untuk menemukan solusi bagi masalah tanpa sensor. Konversi masalah perencanaan tanpa sensor ke masalah perencanaan status keyakinan bekerja dengan cara yang hampir sama, perbedaan utamanya adalah bahwa model transisi fisik yang mendasarinya direpresentasikan oleh kumpulan skema tindakan dan status keyakinan dapat direpresentasikan oleh rumus logika alih-alih serangkaian status yang disebutkan secara eksplisit. Untuk menyederhanakannya, kami berasumsi bahwa masalah perencanaan yang mendasarinya bersifat deterministik.

Status keyakinan awal untuk masalah pengecatan tanpa sensor dapat mengabaikan fluent *InView* karena agen tidak memiliki sensor. Lebih jauh, kami menganggap fakta yang tidak berubah $Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$ sebagai fakta yang diberikan karena fakta-fakta ini berlaku di setiap status keyakinan. Agen tidak mengetahui warna kaleng atau objek, atau apakah kaleng terbuka atau tertutup, tetapi mengetahui bahwa objek dan kaleng memiliki warna: $\forall x \exists c Color(x, c)$. Setelah melakukan Skolemisasi, kita memperoleh status keyakinan awal:

$$b_0 = Color(x, C(x)) .$$

Dalam perencanaan klasik, di mana asumsi dunia tertutup dibuat, kita akan berasumsi bahwa setiap fluent yang tidak disebutkan dalam suatu keadaan adalah salah, tetapi dalam perencanaan tanpa sensor (dan sebagian dapat diamati) kita harus beralih ke asumsi dunia terbuka di mana keadaan mengandung fluent positif dan negatif, dan jika fluent tidak muncul, nilainya tidak diketahui. Dengan demikian, keadaan keyakinan sesuai persis dengan kumpulan kemungkinan dunia yang memenuhi rumus. Mengingat keadaan keyakinan awal ini, urutan tindakan berikut adalah solusinya:

$$[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)] .$$

Sekarang kami tunjukkan cara mengembangkan status keyakinan melalui urutan tindakan untuk menunjukkan bahwa status keyakinan akhir memenuhi tujuan.

Pertama, perhatikan bahwa dalam status keyakinan b tertentu, agen dapat mempertimbangkan tindakan apa pun yang prasyaratnya dipenuhi oleh b . (Tindakan lain tidak dapat digunakan karena model transisi tidak mendefinisikan efek tindakan yang prasyaratnya mungkin tidak terpenuhi.) Rumus umum untuk memperbarui status keyakinan b yang diberikan tindakan yang berlaku a dalam dunia deterministik adalah sebagai berikut:

$$b' = \text{RESULT}(b, a) = \{s' : s' = \text{RESULT}_p(s, a) \text{ and } s \in b\}$$

di mana RESULT_p mendefinisikan model transisi fisik. Untuk saat ini, kami berasumsi bahwa status keyakinan awal selalu merupakan konjungsi literal, yaitu, rumus 1-CNF. Untuk membangun status keyakinan baru b' , kita harus mempertimbangkan apa yang terjadi pada setiap literal ℓ di setiap status fisik s di b saat tindakan a diterapkan. Untuk literal yang nilai kebenarannya sudah diketahui di b , nilai kebenaran di b' dihitung dari nilai saat ini dan daftar tambah dan daftar hapus tindakan. (Misalnya, jika ℓ ada di daftar hapus tindakan, maka $\neg\ell$ ditambahkan ke b' .) Bagaimana dengan literal yang nilai kebenarannya tidak diketahui di b ? Ada tiga kasus:

1. Jika tindakan menambahkan ℓ , maka ℓ akan benar di b' terlepas dari nilai awalnya.
2. Jika tindakan menghapus ℓ , maka ℓ akan salah di b' terlepas dari nilai awalnya.
3. Jika tindakan tidak memengaruhi ℓ , maka ℓ akan mempertahankan nilai awalnya (yang tidak diketahui) dan tidak akan muncul di b' .

Oleh karena itu, kita melihat bahwa perhitungan b' hampir identik dengan kasus yang dapat diamati, yang ditentukan oleh Persamaan (1.1):

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a).$$

Kita tidak dapat menggunakan semantik himpunan dengan tepat karena (1) kita harus memastikan bahwa b' tidak mengandung ℓ dan $\neg\ell$, dan (2) atom dapat mengandung variabel yang tidak terikat. Namun, $\text{RESULT}(b, a)$ masih dihitung dengan memulai dengan b , menetapkan atom apa pun yang muncul di $\text{DEL}(a)$ menjadi false, dan menetapkan atom apa pun yang muncul di $\text{ADD}(a)$ menjadi true. Misalnya, jika kita menerapkan $\text{RemoveLid}(\text{Can}_1)$ ke status keyakinan awal b_0 , kita memperoleh

$$b_1 = \text{Color}(x, C(x)) \wedge \text{Open}(\text{Can}_1).$$

Ketika kita menerapkan tindakan $\text{Paint}(\text{Chair}, \text{Can}_1)$, prasyarat $\text{Color}(\text{Can}_1, c)$ dipenuhi oleh literal yang diketahui $\text{Color}(x, C(x))$ dengan pengikatan $\{x/\text{Can}_1, c/C(\text{Can}_1)\}$ dan status keyakinan baru adalah

$$b_2 = \text{Color}(x, C(x)) \wedge \text{Open}(\text{Can}_1) \wedge \text{Color}(\text{Chair}, C(\text{Can}_1)).$$

Terakhir, kita terapkan tindakan $\text{Paint}(\text{Table}, \text{Can}_1)$ untuk mendapatkan

$$b_2 = \text{Color}(x, C(x)) \wedge \text{Open}(\text{Can}_1) \wedge \text{Color}(\text{Chair}, C(\text{Can}_1)) \\ \wedge \text{Color}(\text{Table}, C(\text{Can}_1)).$$

Keadaan keyakinan akhir memenuhi tujuan, $Color(Table, c) \wedge Color(Chair, c)$, dengan variabel c terikat ke $C(Can_1)$.

Analisis sebelumnya mengenai aturan pembaruan telah menunjukkan fakta yang sangat penting: *himpunan kondisi keyakinan yang didefinisikan sebagai gabungan literal bersifat tertutup di bawah pembaruan yang didefinisikan oleh skema tindakan PDDL. Artinya, jika kondisi keyakinan dimulai sebagai gabungan literal, maka setiap pembaruan akan menghasilkan gabungan literal.* Ini berarti bahwa dalam dunia dengan n **fluent** (proposisi yang nilainya dapat berubah), setiap kondisi keyakinan dapat direpresentasikan oleh gabungan literal berukuran $O(n)$. Ini adalah hasil yang sangat melegakan, mengingat ada 2^n kondisi di dunia. Ini menunjukkan bahwa kita dapat merepresentasikan semua himpunan bagian dari 2^n kondisi tersebut yang akan kita butuhkan secara **ringkas**. Terlebih lagi, proses pemeriksaan kondisi keyakinan yang merupakan himpunan bagian atau himpunan induk dari kondisi keyakinan yang telah dikunjungi sebelumnya juga mudah, setidaknya dalam kasus proposisional.

Di balik gambaran yang menyenangkan ini, ada satu kendala: ini hanya berlaku untuk skema tindakan yang memiliki efek yang sama untuk semua kondisi di mana prasyaratnya terpenuhi. Properti inilah yang memungkinkan pemeliharaan representasi **kondisi keyakinan** 1-CNF. Segera setelah efek dapat bergantung pada kondisi, ketergantungan diperkenalkan di antara **fluent** dan properti 1-CNF hilang. Misalkan **fluent**-nya adalah AtL dan AtR untuk lokasi robot, serta $CleanL$ dan $CleanR$ untuk kondisi kotak. Menurut definisi masalah, tindakan *Suck* tidak memiliki prasyarat—ia selalu dapat dilakukan. Kesulitannya adalah efeknya bergantung pada lokasi robot: ketika robot berada di AtL , hasilnya adalah $CleanL$, tetapi ketika berada di AtR , hasilnya adalah $CleanR$. Untuk tindakan semacam itu, skema tindakan kita akan membutuhkan sesuatu yang baru: **efek kondisional**. Ini memiliki sintaks "ketika kondisi: efek", di mana **kondisi** adalah rumus logis untuk dibandingkan dengan kondisi saat ini, dan **efek** adalah rumus yang menggambarkan kondisi yang dihasilkan. Untuk dunia penyedot debu, kita memiliki:

$$Action(Suck, EFFECT: \textbf{when } AtL: CleanL \wedge \textbf{when } AtR: CleanR) .$$

Ketika diterapkan pada kondisi keyakinan awal *True*, kondisi keyakinan yang dihasilkan adalah $(AtL \wedge CleanL) \vee (AtR \wedge CleanR)$, yang tidak lagi dalam bentuk 1-CNF. Secara umum, efek kondisional dapat menimbulkan ketergantungan arbitrer di antara fluent dalam suatu kondisi keyakinan, yang mengarah pada kondisi keyakinan berukuran eksponensial dalam kasus terburuk.

Penting untuk memahami perbedaan antara prasyarat dan efek kondisional. Semua efek kondisional yang kondisinya terpenuhi akan diterapkan efeknya untuk menghasilkan kondisi yang baru; jika tidak ada yang terpenuhi, maka kondisi yang dihasilkan tidak berubah. Di sisi lain, jika suatu prasyarat tidak terpenuhi, maka tindakan tidak dapat diterapkan dan kondisi yang dihasilkan tidak terdefinisi. Dari sudut pandang perencanaan tanpa sensor, lebih

baik memiliki efek kondisional daripada tindakan yang tidak dapat diterapkan. Misalnya, kita dapat membagi *Suck* menjadi dua tindakan dengan efek tanpa syarat sebagai berikut:

$Action(SuckL, PRECOND: AtL; EFFECT: CleanL)$
 $Action(SuckR, PRECOND: AtR; EFFECT: CleanR)$.

Sekarang kita hanya memiliki skema tanpa syarat, jadi semua status keyakinan tetap dalam 1-CNF; sayangnya, kita tidak dapat menentukan penerapan *SuckL* dan *SuckR* dalam status keyakinan awal.

Maka, tampaknya tak terelakkan bahwa masalah-masalah yang tidak sepele akan melibatkan status keyakinan yang tidak menentu, seperti yang ditemui ketika kita mempertimbangkan masalah estimasi status untuk dunia wumpus. Solusi yang disarankan kemudian adalah menggunakan **perkiraan konservatif** terhadap status keyakinan yang tepat; misalnya, status keyakinan dapat tetap dalam 1-CNF jika berisi semua literal yang nilai kebenarannya dapat ditentukan dan memperlakukan semua literal lainnya sebagai tidak diketahui. Meskipun pendekatan ini bagus, karena tidak pernah menghasilkan rencana yang salah, pendekatan ini tidak lengkap karena mungkin tidak dapat menemukan solusi untuk masalah yang tentu melibatkan interaksi di antara literal. Untuk memberikan contoh yang mudah, jika tujuannya adalah agar robot berada di petak yang bersih, maka [*Suck*] adalah solusinya, tetapi agen tanpa sensor yang bersikeras pada status keyakinan 1-CNF tidak akan menemukannya.

Mungkin solusi yang lebih baik adalah mencari urutan tindakan yang menjaga status keyakinan sesederhana mungkin. Misalnya, di dunia vakum tanpa sensor, urutan tindakan [*Right, Suck, Left, Suck*] menghasilkan urutan status keyakinan berikut:

$$\begin{aligned} b_0 &= True \\ b_1 &= AtR \\ b_2 &= AtR \wedge CleanR \\ b_3 &= AtL \wedge CleanR \\ b_4 &= AtL \wedge CleanR \wedge CleanL \end{aligned}$$

Artinya, agen dapat memecahkan masalah sambil mempertahankan status keyakinan 1-CNF, meskipun beberapa urutan (misalnya, yang dimulai dengan *Suck*) berada di luar 1-CNF. Pelajaran umum tidak hilang dari manusia: kita selalu melakukan tindakan kecil (memeriksa waktu, menepuk-nepuk saku untuk memastikan kita memiliki kunci mobil, membaca rambu jalan saat kita menavigasi melalui kota) untuk menghilangkan ketidakpastian dan menjaga status keyakinan kita agar tetap terkendali.

Ada pendekatan lain yang sangat berbeda untuk masalah status keyakinan yang tidak dapat dikelola: jangan repot-repot menghitungnya sama sekali. Misalkan status keyakinan awal adalah b_0 dan kita ingin mengetahui status keyakinan yang dihasilkan dari urutan tindakan $[a_1, \dots, a_m]$. Alih-alih menghitungnya secara eksplisit, cukup nyatakan sebagai " b_0 lalu $[a_1, \dots, a_m]$." Ini adalah representasi yang malas tetapi tidak ambigu dari status keyakinan,

dan cukup ringkas— $O(n + m)$ di mana n adalah ukuran status keyakinan awal (diasumsikan dalam 1-CNF) dan m adalah panjang maksimum dari urutan tindakan. Namun, sebagai representasi status keyakinan, representasi ini memiliki satu kelemahan: menentukan apakah tujuan terpenuhi, atau tindakan dapat diterapkan, mungkin memerlukan banyak komputasi.

Komputasi dapat diimplementasikan sebagai uji konsekuensi: jika A_m merepresentasikan kumpulan aksioma status penerus yang diperlukan untuk menentukan kejadian tindakan $a_1, \dots, a_m$ —seperti yang dijelaskan untuk SATPLAN dan G_m menegaskan bahwa tujuan tersebut benar setelah m langkah, maka rencana tersebut mencapai tujuan jika $b_0 \wedge A_m \models G_m$, yaitu, jika $b_0 \wedge A_m \wedge \neg G_m$ tidak dapat dipenuhi. Dengan penyelesaian SAT modern, mungkin saja hal ini dapat dilakukan jauh lebih cepat daripada menghitung status keyakinan penuh. Misalnya, jika tidak ada tindakan dalam urutan yang memiliki fluent tujuan tertentu dalam daftar penambahannya, pemecah akan segera mendeteksinya. Akan membantu juga jika hasil parsial tentang status keyakinan—misalnya, fluent yang diketahui benar atau salah—disimpan dalam cache untuk menyederhanakan perhitungan selanjutnya.

Bagian terakhir dari teka-teki perencanaan tanpa sensor adalah fungsi heuristik untuk memandu pencarian. Arti dari fungsi heuristik sama dengan perencanaan klasik: estimasi (mungkin dapat diterima) dari biaya untuk mencapai tujuan dari status keyakinan yang diberikan. Dengan status keyakinan, kita memiliki satu fakta tambahan: memecahkan setiap subset dari status keyakinan tentu lebih mudah daripada memecahkan status keyakinan itu sendiri:

$$\text{if } b_1 \subseteq b_2 \text{ then } h^*(b_1) \leq h^*(b_2).$$

Oleh karena itu, setiap heuristik yang dapat diterima yang dihitung untuk suatu subset dapat diterima untuk status kepercayaan itu sendiri. Kandidat yang paling jelas adalah subset tunggal, yaitu, status fisik individual. Kita dapat mengambil kumpulan acak status $s_1, \dots, s_N$ yang berada dalam status kepercayaan b , menerapkan heuristik yang dapat diterima h , dan mengembalikan

$$H(b) = \max\{h(s_1), \dots, h(s_N)\}$$

sebagai estimasi heuristik untuk menyelesaikan b . Kita juga dapat menggunakan grafik perencanaan langsung pada b itu sendiri: jika itu adalah konjungsi literal (1-CNF), cukup tetapkan literal tersebut menjadi lapisan status awal grafik. Jika b tidak dalam 1-CNF, mungkin saja untuk menemukan kumpulan literal yang bersama-sama memerlukan b . Misalnya, jika b dalam bentuk normal disjuntif (DNF), setiap suku rumus DNF adalah konjungsi literal yang memerlukan b dan dapat membentuk lapisan awal grafik perencanaan. Seperti sebelumnya, kita dapat mengambil nilai maksimum heuristik yang diperoleh dari setiap kumpulan literal. Kita juga dapat menggunakan heuristik yang tidak dapat diterima seperti heuristik abaikan-hapus-daftar, yang tampaknya bekerja cukup baik dalam praktik.

Perencanaan kontinjensi

Kita telah melihat bahwa perencanaan kontinjensi—pembuatan rencana dengan percabangan bersyarat berdasarkan persepsi—cocok untuk lingkungan dengan observabilitas parsial, non-determinisme, atau keduanya. Untuk masalah pengecatan yang dapat diamati sebagian dengan aksioma persepsi yang diberikan sebelumnya, satu solusi kontinjensi yang mungkin adalah sebagai berikut:

```
[LookAt(Table), LookAt(Chair ),
  if Color (Table, c) ∧ Color (Chair, c) then NoOp
  else [RemoveLid(Can1), LookAt(Can1), RemoveLid(Can2), LookAt(Can2),
    if Color (Table, c) ∧ Color (can, c) then Paint(Chair, can)
    else if Color (Chair, c) ∧ Color (can, c) then Paint(Table, can)
    else [Paint(Chair, Can1), Paint(Table, Can1)]]]
```

Variabel dalam rencana ini harus dianggap terkuantifikasi secara eksistensial; baris kedua menyatakan bahwa jika ada beberapa warna c yang merupakan warna meja dan kursi, maka agen tidak perlu melakukan apa pun untuk mencapai tujuan. Saat menjalankan rencana ini, agen perencanaan kontingensi dapat mempertahankan status keyakinannya sebagai rumus logis dan mengevaluasi setiap kondisi cabang dengan menentukan apakah status keyakinan tersebut memerlukan rumus kondisi atau negasinya. (Tergantung pada algoritme perencanaan kontingensi untuk memastikan bahwa agen tidak akan pernah berakhir dalam status keyakinan di mana nilai kebenaran rumus kondisi tidak diketahui.) Perhatikan bahwa dengan kondisi tingkat pertama, rumus dapat dipenuhi dengan lebih dari satu cara; misalnya, kondisi $Color (Table, c) \wedge Color (can, c)$ mungkin dipuaskan oleh $\{can/Can_1\}$ dan oleh $\{can/Can_1\}$ jika kedua kaleng memiliki warna yang sama dengan meja. Dalam hal itu, agen dapat memilih substitusi yang memuaskan untuk diterapkan ke pada sisa rencana.

Penghitungan status keyakinan baru setelah tindakan dan persepsi berikutnya dilakukan dalam dua tahap. Tahap pertama menghitung status keyakinan setelah tindakan, sama seperti untuk agen tanpa sensor:

$$\hat{b} = (b - DEL(a)) \cup ADD(a)$$

di mana, seperti sebelumnya, kita telah mengasumsikan status keyakinan yang direpresentasikan sebagai konjungsi literal. Tahap kedua sedikit lebih rumit. Misalkan literal persepsi $p_1, \dots, p_k$ diterima. Seseorang mungkin berpikir bahwa kita hanya perlu menambahkannya ke dalam status keyakinan; pada kenyataannya, kita juga dapat menyimpulkan bahwa prasyarat untuk penginderaan terpenuhi. Sekarang, jika persepsi p memiliki tepat satu aksioma persepsi, $Percept(p, PRECOND: c)$, di mana c adalah konjungsi literal, maka literal tersebut dapat dimasukkan ke dalam status keyakinan bersama dengan p . Di sisi lain, jika p memiliki lebih dari satu aksioma persepsi yang prasyaratnya mungkin berlaku menurut status keyakinan yang diprediksi $\hat{b}$, maka kita harus menambahkan disjungsi prasyarat tersebut. Jelas, ini membawa status keyakinan di luar 1-CNF dan

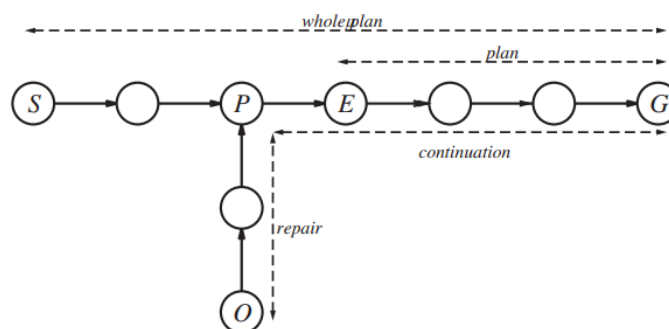
memunculkan komplikasi yang sama seperti efek kondisional, dengan kelas solusi yang hampir sama.

Dengan adanya mekanisme untuk menghitung status keyakinan yang tepat atau perkiraan, kita dapat menghasilkan rencana kontingen dengan perluasan pencarian maju AND–OR atas status keyakinan. Tindakan dengan efek nondeterministik—yang didefinisikan hanya dengan menggunakan disjungsi dalam EFEK skema tindakan—dapat diakomodasi dengan perubahan kecil pada kalkulasi pembaruan status keyakinan dan tanpa perubahan pada algoritme pencarian. Untuk fungsi heuristik, banyak metode yang disarankan untuk perencanaan tanpa sensor juga berlaku dalam kasus nondeterministik yang dapat diamati sebagian.

Perencanaan ulang daring

Bayangkan sedang menonton robot pengelasan titik di pabrik mobil. Gerakan robot yang cepat dan akurat diulang terus-menerus saat setiap mobil melewati jalur produksi. Meskipun secara teknis mengesankan, robot itu mungkin tampak tidak *cerdas* sama sekali karena gerakannya adalah urutan yang tetap dan telah diprogram sebelumnya; robot itu jelas tidak "tahu apa yang dilakukannya" dalam arti yang bermakna. Sekarang, bayangkan sebuah pintu yang terpasang dengan buruk jatuh dari mobil tepat saat robot akan melakukan pengelasan titik. Robot dengan cepat mengganti aktuator pengelasan dengan penjepit, mengambil pintu, memeriksanya dari goresan, memasangnya kembali ke mobil, mengirim email ke supervisor lantai, beralih kembali ke aktuator pengelas, dan melanjutkan pekerjaannya. Tiba-tiba, perilaku robot tersebut tampak memiliki tujuan, bukan sekadar hafalan; kita berasumsi itu bukan hasil dari rencana kontingen yang besar dan telah dihitung sebelumnya, tetapi dari proses perencanaan ulang daring—yang berarti robot tersebut memang perlu tahu apa yang ingin dilakukannya.

Perencanaan ulang mengandaikan adanya semacam **pemantauan eksekusi** untuk menentukan kebutuhan akan rencana baru. Salah satu kebutuhan ini muncul ketika agen perencanaan kontingen lelah merencanakan setiap kemungkinan kecil, seperti apakah langit akan runtuh menimpa kepalanya. Beberapa cabang dari rencana kontingen yang sebagian sudah dibuat dapat secara sederhana menyatakan *Replan*; jika cabang tersebut tercapai selama eksekusi, agen akan kembali ke mode perencanaan. Seperti yang telah disebutkan sebelumnya, keputusan tentang seberapa banyak masalah yang harus diselesaikan di muka dan seberapa banyak yang harus diserahkan kepada perencanaan ulang melibatkan pertimbangan antara peristiwa yang mungkin terjadi dengan biaya dan probabilitas yang berbeda. Tidak ada yang ingin mobilnya mogok di tengah Gurun Sahara dan baru kemudian memikirkan tentang persediaan air yang cukup.



Gambar 2.9 Sebelum eksekusi, perencana membuat rencana, yang di sini disebut rencana menyeluruh, untuk berpindah dari *S* ke *G*. Agen mengeksekusi langkah-langkah rencana hingga ia berharap berada di kondisi *E*, tetapi mengamati bahwa ia sebenarnya berada di *O*. Agen kemudian membuat rencana ulang untuk perbaikan minimal ditambah kelanjutan untuk mencapai *G*.

Perencanaan ulang juga mungkin diperlukan jika model dunia agen tidak tepat. Model untuk suatu tindakan mungkin memiliki prasyarat yang hilang—misalnya, agen mungkin tidak tahu bahwa melepas tutup kaleng cat sering kali memerlukan obeng; model mungkin memiliki efek yang hilang—misalnya, mengecat suatu objek dapat membuat cat juga mengenai lantai; atau model mungkin memiliki variabel status yang hilang—misalnya, model yang diberikan sebelumnya tidak memiliki gagasan tentang jumlah cat dalam kaleng, tentang bagaimana tindakannya memengaruhi jumlah ini, atau tentang perlunya jumlah tersebut menjadi bukan nol. Model mungkin juga tidak memiliki ketentuan untuk kejadian eksogen seperti seseorang menjatuhkan kaleng cat. Kejadian eksogen juga dapat mencakup perubahan dalam tujuan, seperti penambahan persyaratan bahwa meja dan kursi tidak boleh dicat hitam. Tanpa kemampuan untuk memantau dan merencanakan ulang, perilaku agen kemungkinan akan sangat rapuh jika bergantung pada kebenaran mutlak modelnya.

Agen daring memiliki pilihan tentang seberapa cermat memantau lingkungan. Kami membedakan tiga tingkatan:

- **Pemantauan tindakan:** sebelum menjalankan tindakan, agen memverifikasi bahwa semua prasyarat masih berlaku.
- **Pemantauan rencana:** sebelum menjalankan tindakan, agen memverifikasi bahwa rencana yang tersisa akan tetap berhasil.
- **Pemantauan tujuan:** sebelum menjalankan tindakan, agen memeriksa untuk melihat apakah ada serangkaian tujuan yang lebih baik yang dapat dicapainya.

Pada Gambar 2.9 kita melihat skema pemantauan tindakan. Agen melacak rencana awalnya, *keseluruhan rencana*, dan bagian dari rencana yang belum dijalankan, yang dilambangkan dengan rencana. Setelah menjalankan beberapa langkah pertama dari rencana, agen berharap berada dalam keadaan *E*. Namun, agen mengamati bahwa sebenarnya ia berada dalam keadaan *O*. Kemudian, ia perlu memperbaiki rencana dengan menemukan beberapa titik *P* pada rencana awal yang dapat dicapainya kembali. (Mungkin saja

P merupakan keadaan tujuan, G). Agen mencoba meminimalkan total biaya rencana: bagian perbaikan (dari O ke P) ditambah kelanjutan (dari P ke G).

Sekarang mari kita kembali ke contoh masalah untuk mendapatkan kursi dan meja dengan warna yang senada. Misalkan agen tersebut membuat rencana ini:

```
[LookAt(Table), LookAt(Chair),
if Color (Table, c)  $\wedge$  Color (Chair, c) then NoOp
  else [RemoveLid(Can1), LookAt(Can1),
    if Color (Table, c)  $\wedge$  Color (Can1, c) then Paint(Chair, Can1)
    else REPLAN]] .
```

Sekarang agen siap untuk mengeksekusi rencana. Misalkan agen mengamati bahwa meja dan kaleng cat berwarna putih, sedangkan kursi berwarna hitam. Agen kemudian mengeksekusi *Paint(Chair, Can₁)*. Pada titik ini, perencana klasik akan menyatakan kemenangan; rencana telah dieksekusi. Namun, agen pemantauan eksekusi daring perlu memeriksa prasyarat dari sisa rencana yang kosong—bahwa meja dan kursi memiliki warna yang sama. Misalkan agen merasakan bahwa mereka tidak memiliki warna yang sama—faktanya, kursi sekarang berwarna abu-abu belang-belang karena cat hitam masih terlihat. Agen kemudian perlu mencari posisi dalam *whole plan* untuk dituju dan urutan tindakan perbaikan untuk mencapainya. Agen menyadari bahwa kondisi saat ini identik dengan prasyarat sebelum tindakan *Paint(Chair, Can₁)*. sehingga agen memilih urutan kosong untuk perbaikan dan membuat rencananya menjadi urutan [*Paint*] yang sama dengan yang baru saja dicoba. Dengan rencana baru ini, pemantauan eksekusi berlanjut, dan tindakan *Paint* dicoba lagi. Perilaku ini akan berulang hingga kursi terlihat benar-benar dicat. Namun perhatikan bahwa perulangan ini dibuat oleh proses rencana—eksekusi—perencanaan ulang, bukan oleh perulangan eksplisit dalam sebuah rencana. Perhatikan juga bahwa rencana asli tidak perlu mencakup setiap kemungkinan. Jika agen mencapai langkah yang ditandai REPLAN, ia kemudian dapat menghasilkan rencana baru (mungkin melibatkan *Can₂*).

Pemantauan tindakan merupakan metode pemantauan pelaksanaan yang sederhana, tetapi terkadang dapat mengakibatkan perilaku yang kurang cerdas. Misalnya, anggaplah tidak ada cat hitam atau putih, dan agen menyusun rencana untuk memecahkan masalah pengecatan dengan mengecat kursi dan meja dengan warna merah. Anggaplah hanya ada cukup cat merah untuk kursi tersebut. Dengan pemantauan tindakan, agen akan melanjutkan dan mengecat kursi dengan warna merah, kemudian menyadari bahwa kursi tersebut kehabisan cat dan tidak dapat mengecat meja, dan pada saat itu ia akan merencanakan ulang perbaikan—mungkin mengecat kursi dan meja dengan warna hijau. Agen pemantau rencana dapat mendeteksi kegagalan setiap kali keadaan saat ini sedemikian rupa sehingga rencana yang tersisa tidak lagi berfungsi. Dengan demikian, ia tidak akan membuang waktu mengecat kursi dengan warna merah. Pemantauan rencana mencapai hal ini dengan memeriksa prasyarat keberhasilan seluruh rencana yang tersisa—yaitu, prasyarat setiap langkah dalam rencana, kecuali prasyarat yang dicapai oleh langkah lain dalam rencana yang tersisa.

Pemantauan rencana menghentikan pelaksanaan rencana yang gagal sesegera mungkin, alih-alih melanjutkannya hingga kegagalan benar-benar terjadi. Pemantauan rencana juga memungkinkan terjadinya **serendipity**—keberhasilan yang tidak disengaja. Jika seseorang datang dan mengecat meja dengan warna merah pada saat yang sama ketika agen mengecat kursi dengan warna merah, maka prasyarat rencana akhir terpenuhi (tujuan telah tercapai), dan agen dapat pulang lebih awal.

Memodifikasi algoritma perencanaan agar setiap tindakan dalam rencana dianotasi dengan prasyarat tindakan tersebut, sehingga memungkinkan pemantauan tindakan, adalah hal yang mudah. Namun, memungkinkan pemantauan rencana sedikit lebih kompleks. Perencana berurutan parsial dan perencana berbasis graf perencanaan memiliki keuntungan karena mereka telah membangun struktur yang berisi hubungan yang diperlukan untuk pemantauan rencana. Sementara itu, memperkuat perencana ruang kondisi dengan anotasi yang diperlukan dapat dilakukan dengan pencatatan yang cermat saat fluent tujuan diregresikan melalui rencana.

Sekarang setelah kita menjelaskan metode untuk pemantauan dan perencanaan ulang, kita perlu bertanya, "Apakah ini berhasil?" Ini adalah pertanyaan yang sangat sulit. Jika yang kita maksud adalah, "Dapatkah kita menjamin bahwa agen akan selalu mencapai tujuan?" maka jawabannya adalah tidak, karena agen dapat secara tidak sengaja sampai pada jalan buntu yang tidak dapat diperbaiki. Misalnya, agen vakum mungkin memiliki model yang salah dari dirinya sendiri dan tidak tahu bahwa baterainya dapat habis. Begitu baterai habis, ia tidak dapat memperbaiki rencana apa pun. Jika kita mengesampingkan jalan buntu—asumsikan bahwa ada rencana untuk mencapai tujuan dari keadaan apa pun di lingkungan—dan berasumsi bahwa lingkungan tersebut benar-benar nondeterministik, dalam arti bahwa rencana tersebut selalu memiliki peluang keberhasilan pada setiap upaya eksekusi, maka agen pada akhirnya akan mencapai tujuan.

Masalah terjadi ketika suatu tindakan sebenarnya tidak nondeterministik, tetapi lebih bergantung pada beberapa prasyarat yang tidak diketahui agen. Misalnya, terkadang kaleng cat mungkin kosong, jadi mengecat dari kaleng itu tidak akan memberikan efek apa pun. Tidak ada jumlah percobaan ulang yang akan mengubah hal ini. Salah satu solusinya adalah memilih secara acak dari antara serangkaian rencana perbaikan yang memungkinkan, daripada mencoba yang sama setiap saat. Dalam kasus ini, rencana perbaikan dengan membuka kaleng lain mungkin berhasil. Pendekatan yang lebih baik adalah mempelajari model yang lebih baik. Setiap kegagalan prediksi merupakan kesempatan untuk belajar; agen harus mampu memodifikasi model dunianya agar sesuai dengan persepsinya. Sejak saat itu, perencanaan ulang akan mampu menemukan perbaikan yang mengatasi akar masalah, daripada mengandalkan keberuntungan untuk memilih perbaikan yang baik.

2.4 PERENCANAAN MULTIAGEN

Sejauh ini, kita berasumsi bahwa hanya satu agen yang melakukan penginderaan, perencanaan, dan tindakan. Ketika ada beberapa agen di lingkungan, setiap agen menghadapi

masalah perencanaan multiagen di mana ia mencoba mencapai tujuannya sendiri dengan bantuan atau hambatan dari orang lain.

Antara kasus agen tunggal murni dan agen multiagen sejati, terdapat spektrum masalah luas yang menunjukkan berbagai tingkat dekomposisi agen monolitik. Agen dengan banyak efektor yang dapat beroperasi secara bersamaan—misalnya, manusia yang dapat mengetik dan berbicara pada saat yang sama—perlu melakukan perencanaan multieffektor untuk mengelola setiap efektor sambil menangani interaksi positif dan negatif di antara efektor. Ketika efektor secara fisik terpisah menjadi unit-unit yang terpisah—seperti dalam armada robot pengantar di pabrik—perencanaan multieffektor menjadi perencanaan multibodi. Masalah multibodi masih merupakan masalah agen tunggal "standar" selama informasi sensor relevan yang dikumpulkan oleh setiap bodi dapat digabungkan—baik secara terpusat atau di dalam setiap bodi—untuk membentuk perkiraan umum keadaan dunia yang kemudian menginformasikan pelaksanaan rencana keseluruhan; dalam kasus ini, banyak bodi bertindak sebagai satu bodi. Ketika kendala komunikasi membuat ini tidak mungkin, kita memiliki apa yang kadang-kadang disebut masalah perencanaan terdesentralisasi; ini mungkin kurang tepat, karena fase perencanaan terpusat tetapi fase eksekusi setidaknya sebagian terpisah. Dalam kasus ini, sub-rencana yang dibuat untuk setiap bodi mungkin perlu menyertakan tindakan komunikatif eksplisit dengan bodi lain. Misalnya, beberapa robot pengintai yang meliput area luas mungkin sering berada di luar jangkauan kontak radio satu sama lain dan harus berbagi temuan mereka pada saat komunikasi memungkinkan.

Ketika entitas tunggal melakukan perencanaan, sebenarnya hanya ada satu tujuan, yang pasti dimiliki oleh semua tubuh. Ketika tubuh-tubuh tersebut adalah agen-agen berbeda yang melakukan perencanaan sendiri, mereka mungkin masih memiliki tujuan yang sama; misalnya, dua pemain tenis manusia yang membentuk tim ganda berbagi tujuan untuk memenangkan pertandingan. Namun, bahkan dengan tujuan bersama, kasus multibodi dan multiagen sangat berbeda. Dalam tim ganda robot multibodi, satu rencana mendikte tubuh mana yang akan pergi ke mana di lapangan dan tubuh mana yang akan memukul bola. Di sisi lain, dalam tim ganda multiagen, setiap agen memutuskan apa yang harus dilakukan; tanpa metode koordinasi, kedua agen mungkin memutuskan untuk menutupi bagian lapangan yang sama dan masing-masing mungkin menyerahkan bola untuk dipukul oleh yang lain.

Kasus paling jelas dari masalah multiagen, tentu saja, adalah ketika agen memiliki tujuan yang berbeda. Dalam tenis, tujuan dari dua tim yang berlawanan saling bertentangan, yang mengarah ke situasi zero-sum. Penonton dapat dipandang sebagai agen jika dukungan atau penghinaan mereka merupakan faktor yang signifikan dan dapat dipengaruhi oleh perilaku pemain; jika tidak, mereka dapat diperlakukan sebagai aspek alam—seperti Weather—yang diasumsikan tidak peduli dengan niat pemain.

Terakhir, beberapa sistem merupakan campuran dari perencanaan terpusat dan multiagen. Misalnya, perusahaan pengiriman dapat melakukan perencanaan luring terpusat untuk rute truk dan pesawatnya setiap hari, tetapi membiarkan beberapa aspek terbuka untuk keputusan otonom oleh pengemudi dan pilot yang dapat menanggapi situasi lalu lintas dan Weather secara individual. Selain itu, tujuan perusahaan dan karyawannya diselaraskan,

sampai batas tertentu, dengan pembayaran **insentif** (gaji dan bonus)—tanda pasti bahwa ini adalah sistem multiagen sejati.

Perencanaan dengan beberapa tindakan simultan

Untuk saat ini, kita akan memperlakukan pengaturan multieffektor, multibadan, dan multiagen dengan cara yang sama, melabeli mereka secara umum sebagai pengaturan multiaktor, menggunakan istilah umum aktor untuk mencakup efektor, badan, dan agen. Sasaran dari bagian ini adalah untuk mengetahui cara mendefinisikan model transisi, rencana yang benar, dan algoritma perencanaan yang efisien untuk pengaturan multiaktor. Rencana yang benar adalah rencana yang, jika dijalankan oleh para aktor, mencapai tujuan. (Dalam pengaturan multiagen yang sebenarnya, tentu saja, para agen mungkin tidak setuju untuk menjalankan rencana tertentu, tetapi setidaknya mereka *akan* tahu rencana apa yang akan berhasil jika *mereka* setuju untuk menjalankannya.) Untuk kesederhanaan, kita mengasumsikan **sinkronisasi** yang sempurna: setiap tindakan membutuhkan waktu yang sama dan tindakan di setiap titik dalam rencana gabungan bersifat simultan.

```

Actors(A, B)
Init(At(A, LeftBaseline)  $\wedge$  At(B, RightNet)  $\wedge$ 
    Approaching(Ball, RightBaseline)  $\wedge$  Partner (A, B)  $\wedge$  Partner (B, A)
    Goal(Returned(Ball)  $\wedge$  (At(a, RightNet)  $\vee$  At(a, LeftNet)))
    Action(Hit(actor, Ball),
        PRECOND:Approaching(Ball, loc)  $\wedge$  At(actor, loc)
        EFFECT:Returned(Ball))
    Action(Go(actor, to),
        PRECOND:At(actor, loc)  $\wedge$  to  $\neq$  loc,
        EFFECT:At(actor, to)  $\wedge$   $\neg$  At(actor, loc))

```

Gambar 2.10 Masalah tenis ganda. Dua aktor *A* dan *B* bermain bersama dan dapat berada di salah satu dari empat lokasi: *LeftBaseline*, *RightBaseline*, *LeftNet*, dan *RightNet*. Bola dapat dikembalikan hanya jika pemain berada di tempat yang tepat. Perhatikan bahwa setiap tindakan harus menyertakan aktor sebagai argumen.

Kita mulai dengan model transisi; untuk kasus deterministik, ini adalah fungsi $\text{RESULT}(s, a)$. Dalam pengaturan agen tunggal, mungkin ada b pilihan yang berbeda untuk tindakan tersebut; b bisa jadi cukup besar, terutama untuk representasi orde pertama dengan banyak objek untuk ditindaklanjuti, tetapi skema tindakan tetap menyediakan representasi yang ringkas. Dalam pengaturan multiaktor dengan n aktor, tindakan tunggal a digantikan oleh tindakan bersama $\langle a_1, \dots, a_n \rangle$, di mana a_i adalah tindakan yang diambil oleh aktor ke i . Langsung saja, kita melihat dua masalah: pertama, kita harus menggambarkan model transisi untuk b^n tindakan gabungan yang berbeda; kedua, kita memiliki masalah perencanaan gabungan dengan faktor percabangan b^n .

Setelah menggabungkan para aktor ke dalam sistem multiaktor yang memiliki faktor percabangan yang sangat besar, fokus utama penelitian dalam perencanaan multiaktor adalah

memisahkan aktor sejauh mungkin. Tujuannya adalah agar kompleksitas masalah tumbuh secara linier dengan n (jumlah aktor) daripada secara eksponensial. Jika para aktor tidak memiliki interaksi satu sama lain—misalnya, n aktor masing-masing bermain *solitaire*—maka kita dapat dengan mudah menyelesaikan n masalah terpisah. Pertanyaan utamanya adalah, jika para aktor terkopel secara longgar, bisakah kita mencapai sesuatu yang mendekati peningkatan efisiensi eksponensial ini? Ini adalah pertanyaan sentral di banyak bidang AI. Kita telah melihatnya secara eksplisit dalam konteks CSPs, di mana grafik batasan "*tree like*" menghasilkan metode solusi yang efisien. Konsep ini juga muncul dalam konteks basis data pola terpisah dan heuristik aditif untuk perencanaan.

Pendekatan standar untuk masalah yang digabungkan secara longgar adalah dengan berpura-pura bahwa masalah tersebut benar-benar dipisahkan dan kemudian memperbaiki interaksinya. Untuk model transisi, ini berarti menulis skema tindakan seolah-olah para aktor bertindak secara independen. Mari kita lihat bagaimana ini bekerja untuk masalah tenis ganda. Mari kita asumsikan bahwa pada satu titik dalam permainan, tim memiliki tujuan untuk mengembalikan bola yang telah dipukul kepada mereka dan memastikan bahwa setidaknya satu dari mereka menutupi net. Tahap pertama dari definisi multiaktor mungkin terlihat seperti Gambar 2.10. Dengan definisi ini, mudah untuk melihat bahwa rencana bersama berikut ini berhasil:

PLAN 1:

$A : [Go(A, RightBaseline), Hit(A, Ball)]$
 $B : [NoOp(B), NoOp(B)] .$

Namun, masalah muncul ketika sebuah rencana membuat kedua agen memukul bola pada saat yang bersamaan. Di dunia nyata, ini tidak akan berhasil, tetapi skema tindakan untuk "*Hit*" menyatakan bahwa bola akan dikembalikan dengan sukses. Secara teknis, kesulitannya adalah prasyarat membatasi keadaan di mana suatu tindakan dapat dieksekusi dengan sukses, tetapi tidak membatasi tindakan lain yang mungkin mengacaukannya. Kami mengatasi ini dengan memperluas skema tindakan dengan satu fitur baru: daftar tindakan bersamaan yang menyatakan tindakan mana yang harus atau tidak boleh dieksekusi secara bersamaan. Misalnya, tindakan "*Hit*" dapat dijelaskan sebagai berikut:

$Action(Hit(a, Ball),$
 $CONCURRENT: b \neq a \Rightarrow \neg Hit(b, Ball)$
 $PRECOND: Approaching(Ball, loc) \wedge At(a, loc)$
 $EFFECT: Returned(Ball)) .$

Dengan kata lain, Tindakan *Hit* hanya akan menghasilkan efek yang diinginkan jika tidak ada tindakan Memukul lain oleh agen berbeda yang terjadi pada waktu bersamaan. (Dalam pendekatan SATPLAN, hal ini akan ditangani oleh aksioma pengecualian tindakan parsial). Namun, untuk beberapa tindakan, efek yang diinginkan justru hanya tercapai ketika tindakan

lain terjadi secara bersamaan. Sebagai contoh, dibutuhkan dua agen untuk membawa sebuah *cooler* penuh minuman ke lapangan tenis:

Action(Carry(a, cooler, here, there),
 CONCURRENT: $b \neq a \wedge \text{Carry}(b, cooler, here, there)$
 PRECOND: $\text{At}(a, here) \wedge \text{At}(cooler, here) \wedge \text{Cooler}(cooler)$
 EFFECT: $\text{At}(a, there) \wedge \text{At}(cooler, there) \wedge \neg \text{At}(a, here) \wedge \neg \text{At}(cooler, here)$).

Dengan skema tindakan semacam ini, semua algoritme perencanaan yang dijelaskan dalam Bab 11 dapat diadaptasi hanya dengan sedikit modifikasi untuk menghasilkan rencana multiaktor. Sejauh kopling di antara subrencana longgar—yang berarti kendala konkurensi hanya berlaku jarang selama pencarian rencana—seseorang akan berharap berbagai heuristik yang diturunkan untuk perencanaan agen tunggal juga efektif dalam konteks multiaktor. Kita dapat memperluas pendekatan ini dengan penyempurnaan dari dua bab terakhir—HTN, observabilitas parsial, kondisional, pemantauan eksekusi, dan perencanaan ulang—tetapi itu berada di luar cakupan buku ini.

Perencanaan dengan beberapa agen: Kerja sama dan koordinasi

Sekarang, mari kita pertimbangkan pengaturan multiagen sejati, di mana setiap agen membuat rencananya sendiri. Sebagai permulaan, mari kita asumsikan bahwa tujuan dan basis pengetahuan dibagikan. Mungkin Anda berpikir bahwa ini akan mereduksi masalah menjadi kasus multibodi—di mana setiap agen cukup menghitung solusi gabungan dan mengeksekusi bagiannya sendiri dari solusi tersebut. Sayangnya, kata "*solusi*" dalam "*solusi gabungan*" bisa menyesatkan. Untuk tim ganda kita, ada lebih dari satu solusi gabungan yang mungkin:

PLAN 2:

$A : [\text{Go}(A, \text{LeftNet}), \text{NoOp}(A)]$
 $B : [\text{Go}(B, \text{RightBaseline}), \text{Hit}(B, \text{Ball})]$.

Jika kedua agen dapat menyetujui rencana 1 atau rencana 2, tujuan akan tercapai. Namun, jika *A* memilih rencana 2 dan *B* memilih rencana 1, maka tidak akan ada yang mengembalikan bola. Sebaliknya, jika *A* memilih 1 dan *B* memilih 2, maka mereka berdua akan mencoba memukul bola. Para agen mungkin menyadari hal ini, tetapi bagaimana mereka dapat berkoordinasi untuk memastikan bahwa mereka menyetujui rencana tersebut?

Salah satu pilihan untuk mengatasi masalah koordinasi dalam perencanaan multiagen adalah dengan mengadopsi **konvensi** sebelum terlibat dalam aktivitas bersama. Konvensi adalah segala batasan dalam pemilihan rencana bersama. Misalnya, konvensi "tetap di sisi lapanganmu" akan meniadakan rencana 1, sehingga pasangan ganda akan memilih rencana 2. Para pengemudi di jalan menghadapi masalah agar tidak bertabrakan satu sama lain; ini (sebagian) diselesaikan dengan mengadopsi konvensi "tetap di sisi kanan jalan" di sebagian besar negara; alternatifnya, "tetap di sisi kiri," bekerja sama baiknya selama semua agen dalam

suatu lingkungan menyetujuinya. Pertimbangan serupa berlaku untuk pengembangan bahasa manusia, di mana yang penting bukanlah bahasa apa yang harus diucapkan setiap individu, melainkan fakta bahwa suatu komunitas semuanya berbicara bahasa yang sama. Ketika konvensi tersebar luas, mereka disebut hukum sosial.

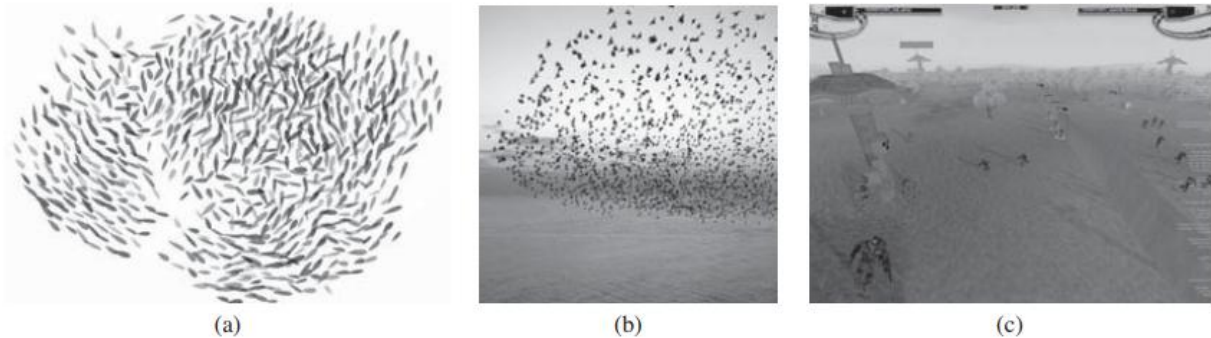
Apabila tidak ada konvensi yang disepakati, para agen dapat memanfaatkan komunikasi untuk mencapai pemahaman bersama tentang rencana gabungan yang bisa diterapkan. Misalnya, seorang pemain tenis bisa berteriak "Saya!" atau "Anda!" untuk menunjukkan rencana gabungan yang diinginkannya. Contohnya, seorang pemain bisa mengomunikasikan rencana gabungan yang diinginkan kepada pemain lain hanya dengan mengeksekusi bagian awal dari rencana tersebut. Jika agen *A* bergerak menuju net, maka agen *B* berkewajiban untuk kembali ke garis belakang untuk memukul bola, karena rencana 2 adalah satu-satunya rencana gabungan yang diawali dengan pergerakan *A*'s menuju net. Pendekatan koordinasi ini, yang kadang disebut pengenalan rencana, berhasil ketika satu tindakan (atau urutan tindakan pendek) cukup untuk menentukan rencana gabungan tanpa keraguan. Penting untuk dicatat bahwa komunikasi dapat bekerja sama baiknya dengan agen yang saling bersaing maupun agen yang saling bekerja sama.

Konvensi juga bisa muncul melalui proses evolusi. Contoh menariknya adalah semut pemanen biji, makhluk sosial yang berevolusi dari tawon yang kurang sosial. Koloni semut mampu menjalankan rencana gabungan yang sangat rumit tanpa kendali terpusat—tugas ratu adalah bereproduksi, bukan melakukan perencanaan terpusat—dan dengan kemampuan komputasi, komunikasi, serta memori yang sangat terbatas pada setiap semut. Koloni semut memiliki banyak peran, termasuk pekerja internal, patroli, dan pencari makan (*foragers*). Setiap semut memilih peran berdasarkan kondisi lokal yang diamatinya. Misalnya, semut pencari makan pergi jauh dari sarang, mencari biji, dan ketika menemukannya, segera membawanya kembali. Dengan demikian, tingkat kembalinya semut pencari makan ke sarang menjadi perkiraan ketersediaan makanan pada hari itu. Ketika tingkat ini tinggi, semut-semut lain akan meninggalkan peran mereka saat ini dan mengambil peran sebagai pencari bangkai (*scavenger*). Semut-semut ini tampaknya memiliki konvensi mengenai pentingnya peran—di mana mencari makan adalah yang paling penting—dan semut akan dengan mudah beralih ke peran yang lebih penting, tetapi tidak ke peran yang kurang penting. Ada juga mekanisme pembelajaran: sebuah koloni belajar untuk melakukan tindakan yang lebih berhasil dan hati-hati sepanjang hidupnya yang bisa berlangsung puluhan tahun, meskipun semut individu hanya hidup sekitar satu tahun.

Satu contoh terakhir dari perilaku multiagen kooperatif muncul dalam perilaku berkelompok burung. Kita dapat memperoleh simulasi kawanan yang wajar jika setiap agen burung (kadang-kadang disebut boid) mengamati posisi tetangga terdekatnya dan kemudian memilih arah dan percepatan yang memaksimalkan jumlah tertimbang dari ketiga komponen ini:

1. **Pemisahan (Separation):** Menghindari kerumunan yang terlalu padat dengan tetangga. Ini mencegah tabrakan.

2. **Penyelarasan (Alignment):** Mencocokkan arah dan kecepatan dengan tetangga. Ini membantu menjaga kawanan tetap bergerak sebagai satu kesatuan.
3. **Kohesi (Cohesion):** Bergerak menuju pusat massa tetangga. Ini menjaga kawanan tetap menyatu dan tidak menyebar.



Gambar 2.11 (a) Kawanan burung simulasi, menggunakan model boids Reynolds. Gambar milik Giuseppe Randazzo, novastructura.net. **(b)** Kawanan burung jalak sungguhan. Gambar oleh Eduardo (pastaboy tidur di flickr). **(c)** Dua tim agen yang bersaing berusaha merebut menara dalam permainan NERO. Gambar milik Risto Miikkulainen.

Jika semua boid menjalankan kebijakan ini, kawanan menunjukkan perilaku terbang yang muncul sebagai benda kaku semu dengan kepadatan yang hampir konstan yang tidak menyebar seiring waktu, dan yang terkadang membuat gerakan menukik tiba-tiba. Anda dapat melihat gambar diam pada Gambar 2.11(a) dan membandingkannya dengan kawanan sebenarnya pada (b). Seperti halnya semut, setiap agen tidak perlu memiliki rencana bersama yang memodelkan tindakan agen lain.

Masalah multiagen yang paling sulit melibatkan kerja sama dengan anggota tim sendiri dan persaingan melawan anggota tim lawan, semuanya tanpa kendali terpusat. Kita melihat ini dalam permainan seperti sepak bola robotik atau permainan NERO yang ditunjukkan pada Gambar 2.11(c), di mana dua tim agen perangkat lunak bersaing untuk merebut menara kontrol. Hingga saat ini, metode untuk perencanaan yang efisien dalam lingkungan seperti ini—misalnya, memanfaatkan kopling longgar—masih dalam tahap awal.

2.5 RINGKASAN

Bab ini telah membahas beberapa komplikasi perencanaan dan tindakan di dunia nyata. Poin-poin utamanya:

- Banyak tindakan mengonsumsi **sumber daya**, seperti uang, bensin, atau bahan mentah. Sangat mudah untuk memperlakukan sumber daya ini sebagai ukuran numerik dalam sebuah kumpulan daripada mencoba bernalar tentang, katakanlah, setiap koin dan uang kertas individual di dunia. Tindakan dapat menghasilkan dan mengonsumsi sumber daya, dan biasanya murah serta efektif untuk memeriksa rencana parsial demi kepatuhan terhadap kendala sumber daya sebelum mencoba penyempurnaan lebih lanjut.

- Waktu adalah salah satu sumber daya yang paling penting. Waktu dapat ditangani oleh algoritme penjadwalan khusus, atau penjadwalan dapat diintegrasikan dengan perencanaan.
- Perencanaan jaringan tugas hierarkis (HTN) memungkinkan agen untuk menerima saran dari perancang domain dalam bentuk **tindakan tingkat tinggi** (HLA) yang dapat diimplementasikan dengan berbagai cara melalui urutan tindakan tingkat rendah. Efek HLA dapat didefinisikan dengan **semantik malaikat**, yang memungkinkan rencana tingkat tinggi yang terbukti benar untuk diturunkan tanpa mempertimbangkan implementasi tingkat rendah. Metode HTN dapat membuat rencana yang sangat besar yang dibutuhkan oleh banyak aplikasi dunia nyata.
- Algoritma perencanaan standar mengasumsikan informasi yang lengkap dan benar serta lingkungan yang deterministik dan dapat diamati sepenuhnya. Banyak domain yang melanggar asumsi ini.
- **Rencana kontingen** memungkinkan agen untuk merasakan dunia selama pelaksanaan untuk memutuskan cabang rencana mana yang akan diikuti. Dalam beberapa kasus, **perencanaan tanpa sensor** atau **perencanaan konforman** dapat digunakan untuk menyusun rencana yang berfungsi tanpa memerlukan persepsi. Baik rencana konforman maupun kontingen dapat disusun dengan pencarian dalam **ruang keadaan keyakinan**. Representasi atau komputasi keadaan keyakinan yang efisien adalah masalah utama.
- Agen perencanaan daring menggunakan pemantauan pelaksanaan dan penyambungan dalam perbaikan sebagaimana diperlukan untuk memulihkan dari situasi yang tidak terduga, yang dapat disebabkan oleh tindakan yang tidak deterministik, kejadian eksogen, atau model lingkungan yang salah.
- Perencanaan multiagen diperlukan ketika ada agen lain di lingkungan yang dapat diajak bekerja sama atau bersaing. Rencana bersama dapat dibuat, tetapi harus ditambah dengan beberapa bentuk koordinasi jika dua agen harus menyetujui rencana bersama mana yang akan dijalankan.
- Bab ini memperluas perencanaan klasik untuk mencakup lingkungan yang tidak deterministik (di mana hasil tindakan tidak pasti), tetapi ini bukanlah kata terakhir tentang perencanaan. Bab 17 menjelaskan teknik untuk lingkungan stokastik (di mana hasil tindakan memiliki probabilitas yang terkait dengannya): proses keputusan Markov, proses keputusan Markov yang dapat diamati sebagian, dan teori permainan. Dalam Bab 21 kami menunjukkan bahwa pembelajaran penguatan memungkinkan agen untuk belajar bagaimana berperilaku dari keberhasilan dan kegagalan masa lalu.

BAB 3

REPRESENTASI PENGETAHUAN

Dalam bab ini, kita akan menunjukkan bagaimana menggunakan logika orde pertama (*first-order logic*) untuk merepresentasikan aspek-aspek terpenting dari dunia nyata, seperti tindakan, ruang, waktu, pikiran, dan kegiatan berbelanja.

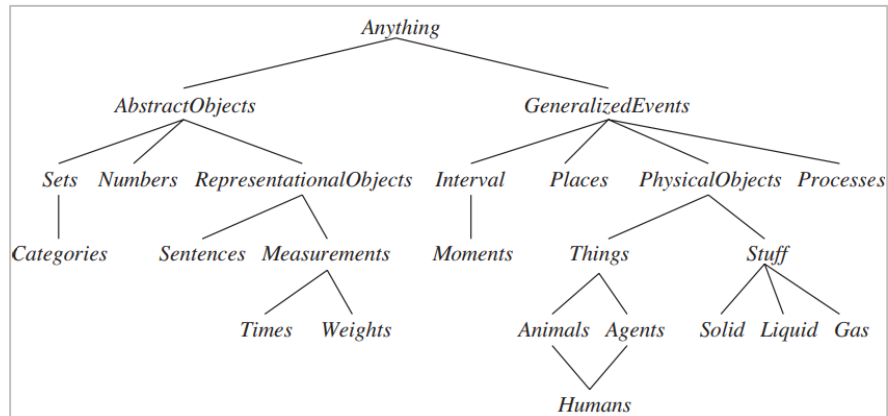
Bab-bab sebelumnya menjelaskan teknologi untuk agen berbasis pengetahuan: sintaksis, semantik, dan teori pembuktian logika proposisional dan tingkat pertama, dan implementasi agen yang menggunakan logika ini. Dalam bab ini kami membahas pertanyaan tentang konten apa yang akan dimasukkan ke dalam basis pengetahuan agen tersebut—bagaimana merepresentasikan fakta tentang dunia.

3.1 REKAYASA ONTOLOGI

Dalam domain "mainan", pilihan representasi tidak terlalu penting; banyak pilihan akan berhasil. Domain kompleks seperti belanja di Internet atau mengemudi mobil dalam lalu lintas membutuhkan representasi yang lebih umum dan fleksibel. Bab ini menunjukkan cara membuat representasi tersebut, berkonsentrasi pada konsep umum—seperti *Events* (Peristiwa), *Time* (Waktu), *Physical Objects* (Objek Fisik), dan *Beliefs* (Keyakinan)—yang muncul di banyak domain berbeda. Merepresentasikan konsep-konsep abstrak ini terkadang disebut **rekayasa ontologis** (*ontological engineering*).

Prospek untuk merepresentasikan segala sesuatu di dunia ini memang sangat menakutkan. Tentu saja, kita tidak akan benar-benar menulis deskripsi lengkap tentang segalanya tetapi kita akan meninggalkan tempat kosong di mana pengetahuan baru untuk domain apa pun dapat dimasukkan.

Misalnya, kita akan mendefinisikan apa artinya menjadi objek fisik, dan detail berbagai jenis objek—robot, televisi, buku, atau apa pun—dapat diisi nanti. Ini analog dengan cara perancang kerangka kerja pemrograman berorientasi objek (seperti kerangka kerja grafis Java Swing) mendefinisikan konsep umum seperti Jendela (*Window*), mengharapakan pengguna untuk menggunakan ini guna mendefinisikan konsep yang lebih spesifik seperti *Jendela Spreadsheet* (*SpreadsheetWindow*). Kerangka kerja konsep umum ini disebut **ontologi atas** (*upper ontology*) karena konvensi menggambar grafik dengan konsep umum di bagian atas dan konsep yang lebih spesifik di bawahnya, seperti pada Gambar 3.1.



Gambar 3.1 menampilkan ontologi atas dunia, yang menunjukkan topik-topik yang akan dibahas nanti dalam bab ini. Setiap tautan menunjukkan bahwa konsep yang lebih rendah adalah spesialisasi dari konsep di atasnya. Spesialisasi tidak harus terpisah; misalnya, manusia adalah hewan sekaligus agen.

Sebelum mempertimbangkan ontologi lebih jauh, kita harus menyatakan satu peringatan penting. Kami telah memilih untuk menggunakan logika orde pertama (*first-order logic* - FOL) untuk membahas konten dan organisasi pengetahuan, meskipun aspek-aspek tertentu dari dunia nyata sulit ditangkap dalam FOL. Kesulitan utamanya adalah bahwa sebagian besar generalisasi memiliki pengecualian atau hanya berlaku sampai tingkat tertentu.

Misalnya, meskipun "tomat berwarna merah" adalah aturan yang berguna, beberapa tomat berwarna hijau, kuning, atau oranye. Pengecualian serupa dapat ditemukan pada hampir semua aturan dalam bab ini. Kemampuan untuk menangani pengecualian dan ketidakpastian (*uncertainty*) sangatlah penting, tetapi ini tidak berhubungan langsung dengan tugas memahami ontologi umum.

Apakah gunanya sebuah ontologi atas? Ontologi tersebut membuat banyak asumsi yang menyederhanakan: waktu dihilangkan sepenuhnya; sinyal tetap dan tidak merambat; struktur sirkuit tetap konstan. Sebuah ontologi yang lebih umum akan mempertimbangkan sinyal pada waktu-waktu tertentu, dan akan mencakup panjang kabel dan penundaan perambatan. Ini akan memungkinkan kita untuk mensimulasikan properti waktu sirkuit, dan memang simulasi semacam itu sering dilakukan oleh perancang sirkuit. Kita juga bisa memperkenalkan kelas gerbang yang lebih menarik, misalnya, dengan menjelaskan teknologi (TTL, CMOS, dan sebagainya) serta spesifikasi input-output. Jika kita ingin membahas keandalan atau diagnosis, kita akan menyertakan kemungkinan bahwa struktur sirkuit atau properti gerbang dapat berubah secara spontan. Untuk memperhitungkan kapasitansi liar, kita perlu merepresentasikan di mana kabel-kabel berada di papan sirkuit.

Jika kita melihat dunia Wumpus, pertimbangan serupa berlaku. Meskipun kita merepresentasikan waktu, ia memiliki struktur yang sederhana: Tidak ada yang terjadi kecuali saat agen bertindak, dan semua perubahan berlangsung instan. Sebuah ontologi yang lebih umum, yang lebih sesuai untuk dunia nyata, akan memungkinkan perubahan simultan yang berlangsung seiring waktu. Kita juga menggunakan predikat Lubang untuk menyatakan kotak

mana yang memiliki lubang. Kita bisa saja memungkinkan jenis lubang yang berbeda dengan memiliki beberapa individu yang termasuk dalam kelas lubang, masing-masing memiliki properti yang berbeda. Demikian pula, kita mungkin ingin mengizinkan hewan lain selain wumpus. Mungkin tidak mungkin untuk menentukan spesies yang tepat dari persepsi yang tersedia, jadi kita perlu membangun taksonomi biologis untuk membantu agen memprediksi perilaku penghuni gua dari petunjuk yang minim.

Untuk setiap ontologi khusus, perubahan seperti ini dapat dilakukan untuk bergerak menuju generalisasi yang lebih besar. Pertanyaan yang jelas kemudian muncul: apakah semua ontologi ini akan berkonvergensi pada satu ontologi tujuan umum? Setelah berabad-abad penyelidikan filosofis dan komputasi, jawabannya adalah "Mungkin." Di bagian ini, kami menyajikan satu ontologi tujuan umum yang mensintesis ide-ide dari berabad-abad tersebut. Dua karakteristik utama ontologi tujuan umum yang membedakannya dari kumpulan ontologi tujuan khusus adalah:

- Ontologi tujuan umum harus dapat diterapkan di lebih banyak atau lebih sedikit domain tujuan khusus (dengan penambahan aksioma khusus domain). Ini berarti tidak ada masalah representasional yang bisa diabaikan atau disembunyikan.
- Dalam domain yang cukup menuntut, berbagai area pengetahuan harus disatukan, karena penalaran dan pemecahan masalah dapat melibatkan beberapa area secara bersamaan. Sistem perbaikan sirkuit robot, misalnya, perlu melakukan penalaran tentang sirkuit dalam hal konektivitas listrik dan tata letak fisik, serta tentang waktu, baik untuk analisis pewaktuan sirkuit maupun estimasi biaya tenaga kerja. Oleh karena itu, kalimat-kalimat yang menggambarkan waktu harus mampu digabungkan dengan kalimat-kalimat yang menggambarkan tata letak spasial dan harus berfungsi sama baiknya untuk nanodetik dan menit, serta untuk angstrom dan meter.

Perlu disebutkan di awal bahwa upaya rekayasa ontologi umum sejauh ini hanya mencapai keberhasilan terbatas. Tidak ada aplikasi AI terkemuka yang menggunakan ontologi bersama—semuanya menggunakan rekayasa pengetahuan tujuan khusus. Pertimbangan sosial/politik dapat mempersulit pihak-pihak yang bersaing untuk menyepakati suatu ontologi. Seperti yang dikatakan Tom Gruber (2004), "Setiap ontologi adalah *perjanjian—kesepakatan sosial*—di antara orang-orang dengan motif yang sama dalam berbagi." Ketika kepentingan yang bersaing lebih besar daripada motivasi untuk berbagi, tidak akan ada ontologi yang sama. Ontologi yang ada telah dibuat melalui empat jalur:

1. Oleh tim ontologis/logikawan terlatih, yang merancang ontologi dan menulis aksioma. Sistem CYC sebagian besar dibangun dengan cara ini.
2. Dengan mengimpor kategori, atribut, dan nilai dari basis data atau basis data yang sudah ada. DBPEDIA dibangun dengan mengimpor fakta terstruktur dari Wikipedia.
3. Dengan menguraikan dokumen teks dan mengekstraksi informasi dari sana. TEXTRUNNER dibangun dengan membaca korpus besar halaman Web.
4. Dengan mendorong amatir yang tidak terlatih untuk memasukkan pengetahuan umum. Sistem OPENMIND dibangun oleh sukarelawan yang mengusulkan fakta dalam bahasa Inggris.

3.2 KATEGORI DAN OBJEK

Pengelompokan objek ke dalam **kategori** adalah bagian penting dari representasi pengetahuan. Meskipun interaksi dengan dunia terjadi pada tingkat objek individual, *sebagian besar penalaran berlangsung pada tingkat kategori*. Misalnya, seorang pembeli biasanya memiliki tujuan untuk membeli bola basket, *bukan bola basket tertentu* seperti BB_9 . Kategori juga berfungsi untuk membuat prediksi tentang objek setelah diklasifikasikan. Seseorang menyimpulkan keberadaan objek tertentu dari masukan persepsi, menyimpulkan keanggotaan kategori dari properti objek yang dirasakan, dan kemudian menggunakan informasi kategori untuk membuat prediksi tentang objek tersebut. Contohnya, dari kulitnya yang berbintik hijau dan kuning, diameter satu kaki, bentuk bulat telur, daging merah, biji hitam, dan keberadaannya di lorong buah, seseorang dapat menyimpulkan bahwa sebuah objek adalah semangka; dari sini, seseorang menyimpulkan bahwa objek tersebut akan berguna untuk salad buah.

Ada dua pilihan untuk merepresentasikan kategori dalam logika tingkat pertama: predikat dan objek. Artinya, kita dapat menggunakan predikat *Bola Basket* (b), atau kita dapat mewujudkan kategori tersebut sebagai objek, *Bola Basket*. Kita kemudian dapat mengatakan *Anggota* ($b, \text{Bola Basket}$), yang akan kita singkat menjadi $b \in \text{Bola Basket}$, untuk mengatakan bahwa b adalah anggota kategori *bola basket*. Kita katakan *Subset* ($\text{Bola Basket}, \text{Bola}$), disingkat menjadi $\text{Bola Basket} \subset \text{Bola}$, untuk mengatakan bahwa *Bola Basket* adalah subkategori *Bola*. Kita akan menggunakan subkategori, subkelas, dan subset secara bergantian.

Kategori berfungsi untuk mengatur dan menyederhanakan basis pengetahuan melalui pewarisan. Jika kita mengatakan bahwa semua instans kategori *Makanan* dapat dimakan, dan jika kita menyatakan bahwa *Buah* adalah subkelas dari *Makanan* dan *Apel* adalah subkelas dari *Buah*, maka kita dapat menyimpulkan bahwa setiap apel dapat dimakan. Kita mengatakan bahwa apel individu mewarisi sifat dapat dimakan, dalam hal ini dari keanggotaannya dalam kategori *Makanan*.

Hubungan subkelas mengorganisir kategori-kategori menjadi taksonomi, atau hierarki taksonomi. Taksonomi telah digunakan secara eksplisit selama berabad-abad dalam bidang teknis. Taksonomi terbesar seperti itu mengorganisir sekitar 10 juta spesies hidup dan punah, banyak di antaranya adalah kumbang menjadi satu hierarki tunggal; ilmu perpustakaan telah mengembangkan taksonomi dari semua bidang pengetahuan, yang dikodekan sebagai sistem *Dewey Decimal*; dan otoritas pajak serta departemen pemerintah lainnya telah mengembangkan taksonomi yang luas untuk pekerjaan dan produk komersial. Taksonomi juga merupakan aspek penting dari pengetahuan umum akal sehat.

Logika orde pertama memudahkan untuk menyatakan fakta tentang kategori, baik dengan mengaitkan objek dengan kategori atau dengan mengkuantifikasi anggotanya. Berikut adalah beberapa jenis fakta, beserta contoh masing-masing:

- Sebuah objek adalah anggota dari suatu kategori. $BB_9 \in \text{Basketballs}$
- Sebuah kategori adalah subkelas dari kategori lain. $\text{Basketballs} \subset \text{Balls}$

- Semua anggota dari suatu kategori memiliki beberapa properti. $(x \in \text{Basketballs}) \Rightarrow \text{Spherical}(x)$
- Anggota suatu kategori dapat dikenali dari beberapa properti. $\text{Orange}(x) \wedge \text{Round}(x) \wedge \text{Diameter}(x) = 9.5'' \wedge x \in \text{Balls} \Rightarrow x \in \text{Basketballs}$
- Suatu kategori secara keseluruhan memiliki beberapa properti. $\text{Dogs} \in \text{DomesticatedSpecies}$

Perhatikan bahwa karena *Anjing* adalah sebuah kategori dan merupakan anggota dari *SpesiesTernak*, maka yang terakhir haruslah **kategori dari kategori**. Tentu saja ada pengecualian untuk banyak aturan di atas (bola basket yang bocor tidak bulat); kita akan membahas pengecualian ini nanti.

Meskipun hubungan subkelas dan anggota adalah yang paling penting untuk kategori, kita juga ingin dapat menyatakan hubungan antara kategori yang bukan merupakan subkelas satu sama lain. Misalnya, jika kita hanya mengatakan bahwa *Jantan* dan *Betina* adalah subkelas dari *Hewan*, maka kita belum mengatakan bahwa yang jantan tidak bisa menjadi yang betina. Kita mengatakan bahwa dua atau lebih kategori adalah **terpisah** (*disjoint*) jika mereka tidak memiliki anggota yang sama. Dan bahkan jika kita tahu bahwa jantan dan betina adalah terpisah, kita tidak akan tahu bahwa hewan yang bukan jantan pasti betina, kecuali jika kita mengatakan bahwa jantan dan betina membentuk **dekomposisi lengkap** (*exhaustive decomposition*) dari hewan. Dekomposisi lengkap yang terpisah dikenal sebagai **partisi**. Contoh-contoh berikut mengilustrasikan ketiga konsep ini:

Disjoint(*{Animals, Vegetables}*)

ExhaustiveDecomposition(*{Americans, Canadians, Mexicans},*
NorthAmericans)

Partition(*{Males, Females}, Animals*)

(Perhatikan bahwa *ExhaustiveDecomposition* dari *NorthAmericans* bukanlah sebuah *Partition*, karena beberapa orang memiliki kewarganegaraan ganda.) Tiga predikat didefinisikan sebagai berikut:

$\text{Disjoint}(s) \Leftrightarrow (\forall c_1, c_2 \ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow \text{Intersection}(c_1, c_2) = \{\})$

$\text{ExhaustiveDecomposition}(s, c) \Leftrightarrow (\forall i \ i \in c \Leftrightarrow \exists c_2 \ c_2 \in s \wedge i \in c_2)$

$\text{Partition}(s, c) \Leftrightarrow \text{Disjoint}(s) \wedge \text{ExhaustiveDecomposition}(s, c)$.

Kategori juga dapat didefinisikan dengan memberikan kondisi penting (*necessary*) dan cukup (*sufficient*) untuk keanggotaan. Contohnya, seorang bujangan adalah seorang pria dewasa yang belum menikah:

$x \in \text{Bachelors} \Leftrightarrow \text{Unmarried}(x) \wedge x \in \text{Adults} \wedge x \in \text{Males}$.

Komposisi fisik

Gagasan bahwa suatu objek dapat menjadi bagian dari objek lain adalah hal yang lumrah. Hidung seseorang adalah bagian dari kepala, Rumania adalah bagian dari Eropa, dan

bab ini adalah bagian dari buku ini. Kita menggunakan hubungan *BagianDari* (*PartOf*) secara umum untuk mengatakan bahwa satu hal adalah bagian dari yang lain. Objek dapat dikelompokkan ke dalam hierarki *BagianDari*, mengingatkan pada hierarki *Subset*:

PartOf (*Bucharest, Romania*)
PartOf (*Romania, EasternEurope*)
PartOf (*EasternEurope, Europe*)
PartOf (*Europe, Earth*) .

Hubungan **BagianDari** bersifat transitif dan refleksif; yaitu:

$PartOf(x, y) \wedge PartOf(y, z) \Rightarrow PartOf(x, z)$.
 $PartOf(x, x)$.

Oleh karena itu, kita dapat menyimpulkan *BagianDari* (*Bucharest, Bumi*).

Kategori **objek gabungan** sering kali dicirikan oleh hubungan struktural antar bagian. Misalnya, hewan berkaki dua memiliki dua kaki yang menempel pada tubuh:

$Biped(a) \Rightarrow \exists l_1, l_2, b \text{ } Leg(l_1) \wedge Leg(l_2) \wedge Body(b) \wedge$
 $PartOf(l_1, a) \wedge PartOf(l_2, a) \wedge PartOf(b, a) \wedge$
 $Attached(l_1, b) \wedge Attached(l_2, b) \wedge$
 $l_1 = l_2 \wedge [\forall l_3 \text{ } Leg(l_3) \wedge PartOf(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)]$.

Notasi untuk "tepat dua" sedikit canggung; kita terpaksa mengatakan bahwa ada dua kaki, bahwa keduanya tidak sama, dan bahwa jika ada yang mengusulkan kaki ketiga, itu harus sama dengan salah satu dari dua kaki lainnya. Di Bagian selanjutnya, kita akan menjelaskan formalisme yang disebut logika deskripsi yang mempermudah representasi batasan seperti "tepat dua."

Kita dapat mendefinisikan hubungan *PartisiBagian* (*PartPartition*) yang analog dengan hubungan *Partisi* (*Partition*) untuk kategori. Sebuah objek terdiri dari bagian-bagian dalam *PartisiBagian*-nya dan dapat dilihat sebagai memperoleh beberapa properti dari bagian-bagian tersebut. Misalnya, massa suatu objek gabungan adalah jumlah massa bagian-bagiannya. Perhatikan bahwa ini tidak terjadi pada kategori, yang tidak memiliki massa, meskipun elemen-elemennya mungkin memilikinya.

Berguna juga untuk mendefinisikan objek gabungan dengan bagian-bagian tertentu tetapi tanpa struktur khusus. Misalnya, kita mungkin ingin mengatakan, "Apel-apel di dalam tas ini beratnya dua pon." Godaannya adalah menganggap berat ini berasal dari himpunan apel di dalam tas, tetapi ini akan menjadi kesalahan karena himpunan adalah konsep matematika abstrak yang memiliki elemen tetapi tidak memiliki berat. Sebaliknya, kita membutuhkan konsep baru, yang akan kita sebut sebagai *kumpulan* (*bunch*). Misalnya, jika apel-apel tersebut adalah *Apel₁*, *Apel₂*, dan *Apel₃*, maka:

$$BunchOf(\{Apple_1, Apple_2, Apple_3\})$$

menunjukkan objek gabungan dengan tiga apel sebagai bagian-bagiannya (bukan elemen). Kita kemudian dapat menggunakan kumpulan sebagai objek normal, meskipun tidak terstruktur. Perhatikan bahwa $BunchOf(\{x\}) = x$. Selanjutnya, *Kumpulan(Apel)* adalah objek gabungan yang terdiri dari semua apel—jangan bingung dengan *Apel*, kategori atau himpunan dari semua apel.

Kita dapat mendefinisikan *Kumpulan(BunchOf)* berdasarkan hubungan *BagianDari* (*PartOf*). Jelas, setiap elemen dari s adalah bagian dari $BunchOf(s)$

$$\forall x x \in s \Rightarrow PartOf(x, BunchOf(s)).$$

Lebih jauh lagi, $BunchOf(s)$ adalah objek terkecil yang memenuhi kondisi ini. Dengan kata lain, $BunchOf(s)$ harus menjadi bagian dari objek apa pun yang memiliki semua elemen s sebagai bagiannya:

$$\forall y [\forall x x \in s \Rightarrow PartOf(x, y)] \Rightarrow PartOf(BunchOf(s), y).$$

Aksioma-aksioma ini adalah contoh dari teknik umum yang disebut **minimalisasi logis**, yang berarti mendefinisikan suatu objek sebagai yang terkecil yang memenuhi kondisi tertentu.

Jenis Alami

Beberapa kategori memiliki definisi yang ketat: sebuah objek adalah segitiga jika dan hanya jika objek tersebut adalah poligon dengan tiga sisi. Di sisi lain, sebagian besar kategori di dunia nyata tidak memiliki definisi yang jelas; ini disebut **kategori jenis alami**. Misalnya, tomat cenderung berwarna merah kusam; kira-kira berbentuk bulat; dengan lekukan di bagian atas tempat tangkai; berdiameter sekitar dua hingga empat inci; dengan kulit tipis namun liat; serta memiliki daging, biji, dan jus di dalamnya. Namun, ada variasi: beberapa tomat berwarna kuning atau oranye, tomat yang belum matang berwarna hijau, beberapa lebih kecil atau lebih besar dari rata-rata, dan tomat ceri memiliki ukuran yang seragam dan kecil. Daripada memiliki definisi tomat yang lengkap, kita memiliki seperangkat fitur yang berfungsi untuk mengidentifikasi objek-objek yang jelas merupakan tomat yang khas, tetapi mungkin tidak dapat memutuskan untuk objek lain. (Mungkinkah ada tomat yang berbulu seperti buah persik?).

Ini menimbulkan masalah bagi agen logis. Agen tidak bisa yakin bahwa objek yang telah diamatinya adalah tomat, dan bahkan jika ia yakin, ia tidak bisa memastikan properti mana dari tomat pada umumnya yang dimiliki objek ini. Masalah ini merupakan konsekuensi tak terhindarkan dari beroperasi di lingkungan yang dapat diamati sebagian.

Salah satu pendekatan yang berguna adalah memisahkan apa yang benar untuk semua instans kategori dari apa yang hanya benar untuk instans *khas* (*typical*). Jadi, selain kategori *Tomat*, kita juga akan memiliki kategori *Typical(Tomatoes)*. Di sini, fungsi *Khas* memetakan sebuah kategori ke subkelas yang hanya berisi instans khas:

$$\text{Typical}(c) \subseteq c$$

Sebagian besar pengetahuan tentang jenis alami (*natural kinds*) sebenarnya akan berkaitan dengan instans khas (*typical*) mereka:

$$x \in \text{Typical}(\text{Tomatoes}) \Rightarrow \text{Red}(x) \wedge \text{Round}(x).$$

Dengan demikian, kita dapat menuliskan fakta-fakta yang berguna tentang kategori tanpa definisi yang tepat. Kesulitan dalam memberikan definisi yang tepat untuk sebagian besar kategori alami telah dijelaskan secara mendalam oleh Wittgenstein (1953). Ia menggunakan contoh *permainan* untuk menunjukkan bahwa anggota suatu kategori memiliki "kemiripan keluarga" (*family resemblances*) daripada karakteristik yang penting (*necessary*) dan cukup (*sufficient*): definisi ketat apa yang mencakup catur, permainan tag, solitaire, dan dodgeball?

Kegunaan gagasan definisi ketat juga ditentang oleh Quine (1953). Ia menunjukkan bahwa bahkan definisi "bujangan" sebagai pria dewasa yang belum menikah pun patut dipertanyakan; seseorang mungkin, misalnya, mempertanyakan pernyataan seperti "Paus adalah seorang bujangan." Meskipun tidak secara ketat *salah*, penggunaan ini tentu saja *tidak tepat* (*infelicitous*) karena menimbulkan kesimpulan yang tidak dimaksudkan oleh pendengar. Ketegangan ini mungkin dapat diselesaikan dengan membedakan antara definisi logis yang cocok untuk representasi pengetahuan internal dan kriteria yang lebih bernuansa untuk penggunaan linguistik yang tepat. Yang terakhir dapat dicapai dengan "menyaring" pernyataan yang berasal dari yang pertama. Ada juga kemungkinan bahwa kegagalan penggunaan linguistik berfungsi sebagai umpan balik untuk memodifikasi definisi internal, sehingga penyaringan menjadi tidak perlu.

Pengukuran

Dalam teori ilmiah maupun teori akal sehat tentang dunia, objek memiliki tinggi, massa, biaya, dan sebagainya. Nilai-nilai yang kita tetapkan untuk properti ini disebut **ukuran**. Ukuran kuantitatif biasa cukup mudah direpresentasikan. Kita membayangkan bahwa alam semesta mencakup "objek ukuran" abstrak, seperti panjang yang merupakan panjang segmen garis ini:  Kita bisa menyebut panjang ini 1,5 inci atau 3,81 sentimeter. Dengan demikian, panjang yang sama memiliki nama yang berbeda dalam bahasa kita. Kita merepresentasikan panjang dengan fungsi **satuan** yang mengambil angka sebagai argumen. Jika segmen garis disebut L_1 , kita bisa menulis:

$$\text{Length}(L_1) = \text{Inches}(1.5) = \text{Centimeters}(3.81).$$

Konversi antar satuan dilakukan dengan menyamakan kelipatan satu satuan dengan satuan lainnya:

$$\text{Centimeters}(2.54 \times d) = \text{Inches}(d).$$

Aksioma serupa dapat ditulis untuk pound dan kilogram, detik dan hari, serta dolar dan sen. Ukuran dapat digunakan untuk menggambarkan objek sebagai berikut:

$$\text{Diameter}(\text{Basketball } 12) = \text{Inches}(9.5).$$

$$\text{ListPrice}(\text{Basketball } 12) = \$ (19).$$

$$d \in \text{Days} \Rightarrow \text{Duration}(d) = \text{Hours}(24).$$

Perhatikan bahwa \$(1) bukan uang kertas satu dolar! Seseorang bisa memiliki dua lembar uang kertas satu dolar, tetapi hanya ada satu objek yang dinamai \$(1). Perhatikan juga bahwa, meskipun *Inches*(0) dan *Centimeters*(0) merujuk pada panjang nol yang sama, keduanya tidak identik dengan ukuran nol lainnya, seperti *Seconds*(0).

Ukuran kuantitatif sederhana mudah direpresentasikan. Ukuran lain menimbulkan masalah yang lebih besar, karena tidak memiliki skala nilai yang disepakati. Latihan memiliki tingkat kesulitan, hidangan penutup memiliki kelezatan, dan puisi memiliki keindahan, namun angka tidak dapat diberikan pada kualitas-kualitas ini. Seseorang mungkin, dalam momen perhitungan murni, mengabaikan properti tersebut sebagai tidak berguna untuk tujuan penalaran logis; atau, lebih buruk lagi, mencoba memaksakan skala numerik pada keindahan. Ini akan menjadi kesalahan besar, karena tidak perlu. Aspek terpenting dari ukuran bukanlah nilai numerik tertentu, tetapi fakta bahwa ukuran dapat *diurutkan*.

Meskipun ukuran bukanlah angka, kita tetap dapat membandingkannya, menggunakan simbol urutan seperti >. Misalnya, kita mungkin percaya bahwa latihan Norvig lebih sulit daripada latihan Russell, dan bahwa seseorang mendapatkan skor lebih rendah pada latihan yang lebih sulit:

$$\begin{aligned} e_1 \in \text{Exercises} \wedge e_2 \in \text{Exercises} \wedge \text{Wrote}(\text{Norvig}, e_1) \wedge \text{Wrote}(\text{Russell}, e_2) \\ \Rightarrow \text{Difficulty}(e_1) > \text{Difficulty}(e_2). \end{aligned}$$

$$\begin{aligned} e_1 \in \text{Exercises} \wedge e_2 \in \text{Exercises} \wedge \text{Difficulty}(e_1) > \text{Difficulty}(e_2) \\ \Rightarrow \text{ExpectedScore}(e_1) < \text{ExpectedScore}(e_2). \end{aligned}$$

Ini cukup untuk memungkinkan seseorang memutuskan latihan mana yang harus dikerjakan, meskipun tidak ada nilai numerik untuk kesulitan yang pernah digunakan. (Namun, seseorang harus mencari tahu siapa yang menulis latihan yang mana.) Jenis hubungan monoton (*monotonic*) di antara ukuran-ukuran ini membentuk dasar bagi bidang **fisika kualitatif**, subbidang AI yang menyelidiki bagaimana bernalar tentang sistem fisik tanpa harus masuk ke persamaan detail dan simulasi numerik.

Objek: Benda dan barang

Dunia nyata dapat dipandang terdiri dari objek primitif (misalnya, partikel atom) dan objek **gabungan** yang dibangun darinya. Dengan bernalar pada tingkat objek besar seperti apel dan mobil, kita dapat mengatasi kerumitan yang terlibat dalam menangani sejumlah besar objek primitif secara individual. Namun, ada bagian signifikan dari kenyataan yang tampaknya menentang **individuasi** yang jelas—pembagian menjadi objek-objek yang berbeda. Kita

memberikan bagian ini nama umum **materi** (*stuff*). Misalnya, anggaplah saya memiliki mentega dan aardvark di depan saya. Saya bisa mengatakan ada satu aardvark, tetapi tidak ada jumlah "objek mentega" yang jelas, karena setiap bagian dari objek mentega juga merupakan objek mentega, setidaknya sampai kita mencapai bagian yang sangat kecil. Inilah perbedaan utama antara *materi* dan *benda*. Jika kita memotong aardvark menjadi dua, kita tidak mendapatkan dua aardvark (sayangnya).

Bahasa Inggris dengan jelas membedakan antara *materi* (*stuff*) dan *benda* (*things*). Kita mengatakan "seekor aardvark," tetapi, kecuali di restoran-restoran California yang sok, seseorang tidak bisa mengatakan "sebuah mentega." Ahli bahasa membedakan antara **kata benda hitung** (*count nouns*), seperti aardvark, lubang, dan teorema, dan **kata benda massa** (*mass nouns*), seperti mentega, air, dan energi. Beberapa ontologi yang bersaing mengklaim dapat menangani perbedaan ini. Di sini kita hanya akan menjelaskan satu; yang lainnya dibahas di bagian catatan sejarah.

Untuk merepresentasikan *materi* (*stuff*) dengan benar, kita mulai dengan hal yang jelas. Kita perlu memiliki objek dalam ontologi kita setidaknya "gumpalan" besar *materi* yang kita gunakan. Misalnya, kita mungkin mengenali segumpal mentega sebagai yang tertinggal di meja malam sebelumnya; kita bisa mengambilnya, menimbanginya, menjualnya, atau apa pun. Dalam pengertian ini, ia adalah objek sama seperti aardvark. Mari kita sebut itu *Mentega*₃. Kita juga mendefinisikan kategori *Mentega*. Secara informal, elemen-elemennya adalah semua hal yang bisa dikatakan "Ini mentega," termasuk *Mentega*₃. Dengan beberapa peringatan tentang bagian-bagian yang sangat kecil yang untuk saat ini akan kita abaikan, setiap bagian dari objek mentega juga merupakan objek mentega:

$$b \in Butter \wedge PartOf(p, b) \Rightarrow p \in Butter .$$

Sekarang kita dapat mengatakan bahwa mentega meleleh pada suhu sekitar 30 derajat celcius:

$$b \in Butter \Rightarrow MeltingPoint(b, Centigrade(30)) .$$

Kita bisa melanjutkan dengan mengatakan bahwa mentega berwarna kuning, kurang padat dari air, lunak pada suhu kamar, memiliki kandungan lemak tinggi, dan sebagainya. Di sisi lain, mentega tidak memiliki ukuran, bentuk, atau berat tertentu. Kita dapat mendefinisikan kategori mentega yang lebih spesifik seperti *UnsaltedButter*, yang juga merupakan jenis *materi* (*stuff*). Perhatikan bahwa kategori *PoundOfButter*, yang mencakup semua objek mentega seberat satu pon sebagai anggotanya, bukanlah jenis *materi*. Jika kita memotong satu pon mentega menjadi dua, sayangnya kita tidak mendapatkan dua pon mentega.

Yang sebenarnya terjadi adalah ini: beberapa properti bersifat **intrinsik**: properti tersebut melekat pada substansi objek itu sendiri, bukan pada objek secara keseluruhan. Ketika Anda memotong suatu contoh *materi* menjadi dua, kedua bagian tersebut mempertahankan properti intrinsik—hal-hal seperti kepadatan, titik didih, rasa, warna, kepemilikan, dan sebagainya. Di sisi lain, properti **ekstrinsik**-nya—berat, panjang, bentuk, dan sebagainya—tidak

dipertahankan saat terjadi pembagian. Kategori objek yang mencakup dalam definisinya hanya properti *intrinsik*, maka disebut **substansi**, atau **kata benda massa** (*mass noun*); sedangkan kelas yang mencakup properti ekstrinsik *apapun* dalam definisinya disebut kata benda hitung (*count noun*). Kategori *Materi* adalah kategori substansi paling umum, yang tidak menentukan properti intrinsik apa pun. Kategori *Benda* adalah kategori objek diskrit paling umum, yang tidak menentukan properti ekstrinsik apa pun.

3.3 EVENT CALCULUS

Kita menunjukkan bagaimana kalkulus situasi merepresentasikan tindakan dan efeknya. Kalkulus situasi terbatas dalam penerapannya: kalkulus ini dirancang untuk menggambarkan dunia di mana tindakan bersifat diskrit, instan, dan terjadi satu per satu. Pertimbangkan tindakan berkelanjutan, seperti mengisi bak mandi. Kalkulus situasi dapat mengatakan bahwa bak mandi kosong sebelum tindakan dan penuh setelah tindakan selesai, tetapi tidak dapat berbicara tentang apa yang terjadi *selama* tindakan. Kalkulus ini juga tidak dapat menggambarkan dua tindakan yang terjadi pada waktu yang sama—seperti menyikat tooth sambil menunggu bak mandi terisi. Untuk menangani kasus-kasus seperti itu, kami memperkenalkan formalisme alternatif yang dikenal sebagai **kalkulus kejadian** (*event calculus*), yang didasarkan pada titik waktu daripada situasi.

Kalkulus kejadian mereifikasi (*reifies*) fluent dan kejadian (*events*). Fluent *Di(Shankar, Berkeley)* adalah sebuah objek yang mengacu pada fakta Shankar berada di Berkeley, tetapi tidak dengan sendirinya mengatakan apa pun tentang apakah itu benar. Untuk menyatakan bahwa sebuah fluent benar pada suatu titik waktu, kita menggunakan predikat *T*, seperti dalam $T(At(Shankar, Berkeley), t)$

Peristiwa digambarkan sebagai contoh kategori peristiwa. Peristiwa E_1 Shankar yang terbang dari San Francisco ke Washington, D.C. digambarkan sebagai

$$E_1 \in Flyings \wedge Flyer(E_1, Shankar) \wedge Origin(E_1, SF) \wedge Destination(E_1, DC).$$

Jika ini terlalu bertele-tele, kita dapat mendefinisikan versi tiga argumen alternatif dari kategori peristiwa terbang dan mengatakan

$$E_1 \in Flyings(Shankar, SF, DC).$$

Kita kemudian menggunakan *Happens* (E_1, i) untuk mengatakan bahwa peristiwa E_1 terjadi selama interval waktu i , dan kita mengatakan hal yang sama dalam bentuk fungsional dengan *Extent* (E_1) = i . Kita merepresentasikan interval waktu dengan pasangan waktu (awal, akhir); yaitu, $i = (t_1, t_2)$ adalah interval waktu yang dimulai pada t_1 dan berakhir pada t_2 . Kumpulan predikat lengkap untuk satu versi kalkulus peristiwa adalah

$T(f, t)$	Fluent f benar pada waktu t .
$Happens(e, i)$	Kejadian e terjadi selama interval waktu i .

$Initiates(e, f, t)$	Kejadian e menyebabkan fluent f mulai berlaku pada waktu t .
$Terminates(e, f, t)$	Kejadian e menyebabkan fluent f berhenti berlaku pada waktu t .
$Clipped(f, i)$	Fluent f berhenti menjadi benar pada suatu titik selama interval waktu i .
$Restored(f, i)$	Fluent f menjadi benar suatu saat selama interval waktu i .

Kami mengasumsikan adanya kejadian khusus, *Mulai (Start)*, yang menggambarkan keadaan awal dengan menyatakan fluent mana yang dimulai (*initiated*) atau diakhiri (*terminated*) pada waktu mulai. Kami mendefinisikan T dengan mengatakan bahwa sebuah fluent berlaku pada suatu titik waktu jika fluent tersebut dimulai oleh suatu kejadian di masa lalu dan tidak dijadikan salah (terpotong) oleh kejadian yang mengintervensi. Sebuah fluent tidak berlaku jika fluent tersebut diakhiri oleh suatu kejadian dan tidak dijadikan benar (dipulihkan) oleh kejadian lain. Secara formal, aksioma-aksiomanya adalah:

$$\begin{aligned}
 &Happens(e, (t_1, t_2)) \wedge Initiates(e, f, t_1) \wedge \neg Clipped(f, (t_1, t)) \wedge t_1 < t \Rightarrow T(f, t) \\
 &Happens(e, (t_1, t_2)) \wedge Terminates(e, f, t_1) \wedge \neg Restored(f, (t_1, t)) \wedge t_1 < t \\
 &\quad \Rightarrow \neg T(f, t)
 \end{aligned}$$

di mana *Clipped* dan *Restored* didefinisikan oleh

$$\begin{aligned}
 Clipped(f, (t_1, t_2)) &\Leftrightarrow \exists e, t, t_3 \text{ Happens}(e, (t, t_3)) \wedge t_1 \leq t < t_2 \\
 Restored(f, (t_1, t_2)) &\Leftrightarrow \exists e, t, t_3 \text{ Happens}(e, (t, t_3)) \wedge t_1 \leq t \wedge Initiates(e, f, t)
 \end{aligned}$$

Mudah untuk memperluas T agar dapat bekerja pada interval dan titik waktu; suatu fasih berlaku pada suatu interval jika berlaku pada setiap titik dalam interval tersebut:

$$T(f, (t_1, t_2)) \Leftrightarrow [\forall t (t_1 \leq t < t_2) \Rightarrow T(f, t)]$$

Fluent dan tindakan didefinisikan dengan aksioma khusus domain yang mirip dengan aksioma status penerus. Misalnya, kita dapat mengatakan bahwa satu-satunya cara agen dunia wumpus mendapatkan anak panah adalah di awal, dan satu-satunya cara untuk menghabiskan anak panah adalah dengan menembakkannya:

$$\begin{aligned}
 Initiates(e, HaveArrow(a), t) &\Leftrightarrow e = Start \\
 Terminates(e, HaveArrow(a), t) &\Leftrightarrow e \in Shootings(a)
 \end{aligned}$$

Dengan mereifikasi kejadian, kita memungkinkan penambahan informasi arbitrer sebanyak apa pun tentangnya. Misalnya, kita bisa mengatakan bahwa penerbangan Shankar *bergoncang* ($Bumpy(E_1)$). Dalam ontologi di mana kejadian adalah predikat n -ary, tidak ada cara untuk menambahkan informasi ekstra seperti ini; beralih ke predikat $n + 1$ -ary bukanlah solusi yang terukur (*scalable*).

Kita dapat memperluas kalkulus kejadian untuk memungkinkan representasi kejadian simultan (seperti dua orang diperlukan untuk menaiki jungkat-jungkit), kejadian eksogen (seperti angin bertiup dan mengubah lokasi objek), kejadian berkelanjutan (seperti permukaan air di bak mandi yang terus meningkat) dan komplikasi lainnya.

Proses

Kejadian yang telah kita lihat sejauh ini adalah yang kita sebut **kejadian diskrit**—kejadian tersebut memiliki struktur yang pasti. Perjalanan Shankar memiliki awal, tengah, dan akhir. Jika terputus di tengah jalan, kejadian tersebut akan menjadi sesuatu yang berbeda—itu tidak akan menjadi perjalanan dari San Francisco ke Washington, melainkan perjalanan dari San Francisco ke suatu tempat di atas Kansas. Di sisi lain, kategori kejadian yang dilambangkan dengan *Penerbangan* (*Flyings*) memiliki kualitas yang berbeda. Jika kita mengambil interval kecil dari penerbangan Shankar, katakanlah, segmen 20 menit ketiga (saat ia menunggu kacang dengan cemas), kejadian itu masih merupakan anggota *Penerbangan*. Bahkan, ini berlaku untuk setiap sub-interval.

Kategori kejadian dengan properti ini disebut **kategori proses** atau **kategori kejadian cair** (*liquid event categories*). Setiap proses e yang terjadi selama suatu interval juga terjadi selama subinterval apa pun:

$$(e \in \text{Processes}) \wedge \text{Happens}(e, (t_1, t_4)) \wedge (t_1 < t_2 < t_3 < t_4) \Rightarrow \text{Happens}(e, (t_2, t_3)).$$

Perbedaan antara kejadian cair (*liquid*) dan **non-cair** (*nonliquid*) sama persis dengan perbedaan antara substansi atau *materi* (*stuff*), dan objek individual atau *benda* (*things*). Bahkan, beberapa orang menyebut kejadian cair sebagai *substansi temporal*, sedangkan substansi seperti mentega adalah **substansi spasial**.

Interval waktu

Kalkulus kejadian membuka kemungkinan bagi kita untuk berbicara tentang waktu, dan interval waktu. Kita akan mempertimbangkan dua jenis interval waktu: momen dan interval diperpanjang. Perbedaannya adalah bahwa hanya momen yang memiliki durasi nol:

$$\text{Partition}(\{\text{Moments}, \text{ExtendedIntervals}\}, \text{Intervals}) \\ i \in \text{Moments} \Leftrightarrow \text{Duration}(i) = \text{Seconds}(0).$$

Selanjutnya kita menciptakan skala waktu dan mengaitkan titik-titik pada skala tersebut dengan momen, memberi kita waktu absolut. Skala waktu ini arbitrer; kita mengukurnya dalam detik dan mengatakan bahwa momen pada tengah malam (GMT) tanggal 1 Januari 1900, memiliki waktu 0. Fungsi *Mulai* (*Begin*) dan *Akhir* (*End*) memilih momen paling awal dan paling akhir

dalam suatu interval, dan fungsi *Waktu (Time)* memberikan titik pada skala waktu untuk suatu momen. Fungsi *Durasi (Duration)* memberikan perbedaan antara waktu akhir dan waktu mulai.

$$Interval(i) \Rightarrow Duration(i) = (Time(End(i)) - Time(Begin(i))) .$$

$$Time(Begin(AD1900)) = Seconds(0) .$$

$$Time(Begin(AD2001)) = Seconds(3187324800) .$$

$$Time(End(AD2001)) = Seconds(3218860800) .$$

$$Duration(AD2001) = Seconds(31536000) .$$

Agar angka-angka ini lebih mudah dibaca, kami juga memperkenalkan fungsi *Tanggal (Date)*, yang membutuhkan enam argumen (jam, menit, detik, hari, bulan, dan tahun) dan mengembalikan titik waktu:

$$Time(Begin(AD2001)) = Date(0, 0, 0, 1, Jan, 2001)$$

$$Date(0, 20, 21, 24, 1, 1995) = Seconds(3000000000) .$$

Dua interval *Berhimpit (Meet)* jika waktu akhir interval pertama sama dengan waktu mulai interval kedua. Kumpulan lengkap hubungan interval, seperti yang diusulkan oleh Allen (1983), ditunjukkan secara grafis pada Gambar 3.2 dan secara logis di bawah ini:

$$Meet(i, j) \Leftrightarrow End(i) = Begin(j)$$

$$Before(i, j) \Leftrightarrow End(i) < Begin(j)$$

$$After(j, i) \Leftrightarrow Before(i, j)$$

$$During(i, j) \Leftrightarrow Begin(j) < Begin(i) < End(i) < End(j)$$

$$Overlap(i, j) \Leftrightarrow Begin(i) < Begin(j) < End(i) < End(j)$$

$$Begins(i, j) \Leftrightarrow Begin(i) = Begin(j)$$

$$Finishes(i, j) \Leftrightarrow End(i) = End(j)$$

$$Equals(i, j) \Leftrightarrow Begin(i) = Begin(j) \wedge End(i) = End(j)$$

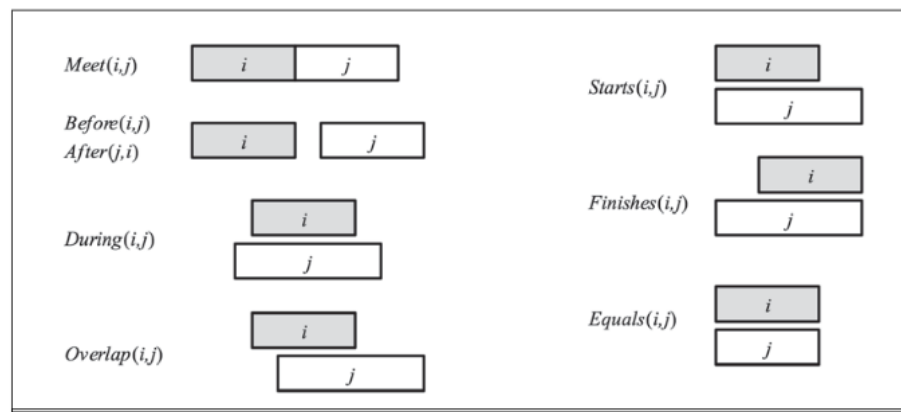
Ini semua memiliki makna intuitifnya, kecuali untuk TumpangTindih (*Overlap*): kita cenderung berpikir tumpang tindih itu simetris (jika *i* tumpang tindih *j* maka *j* tumpang tindih *i*), tetapi dalam definisi ini, *Overlap(i, j)* hanya berlaku jika *i* dimulai sebelum *j*. Untuk mengatakan bahwa masa pemerintahan Elizabeth II segera mengikuti masa pemerintahan George VI, dan masa pemerintahan Elvis tumpang tindih dengan tahun 1950-an, kita dapat menulis yang berikut:

$$Meets(ReignOf(GeorgeVI), ReignOf(ElizabethII)) .$$

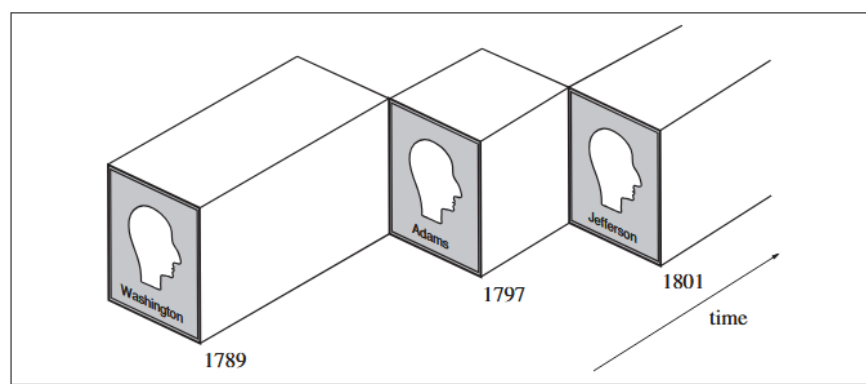
$$Overlap(Fifties, ReignOf(Elvis)) .$$

$$Begin(Fifties) = Begin(AD1950) .$$

$$End(Fifties) = End(AD1959) .$$



Gambar 3.2 Predikat pada interval waktu.



Gambar 3.3 Tampilan skema objek Presiden (USA) selama 15 tahun pertamanya.

Fluent dan objek

Objek fisik dapat dipandang sebagai peristiwa yang digeneralisasi, dalam artian bahwa objek fisik adalah sepotong ruang-waktu. Misalnya, *Amerika Serikat* dapat dianggap sebagai peristiwa yang dimulai pada tahun 1776 sebagai persatuan 13 negara bagian dan masih berlangsung hingga saat ini sebagai persatuan 50 negara bagian. Kita dapat menggambarkan perubahan properti Amerika Serikat menggunakan state fluents, seperti *Populasi(Amerika Serikat)*. Salah satu properti Amerika Serikat yang berubah setiap empat atau delapan tahun, kecuali ada insiden, adalah presidennya. Seseorang mungkin mengusulkan bahwa *Presiden(Amerika Serikat)* adalah istilah logis yang merujuk pada objek yang berbeda pada waktu yang berbeda. Sayangnya, ini tidak mungkin, karena suatu istilah merujuk tepat pada satu objek dalam struktur model tertentu. Istilah *Presiden(Amerika Serikat, t)* dapat merujuk pada objek yang berbeda, tergantung pada nilai t , tetapi ontologi kita memisahkan indeks waktu dari fluents). Satu-satunya kemungkinan adalah bahwa *Presiden(Amerika Serikat)* merujuk pada satu objek tunggal yang terdiri dari orang yang berbeda pada waktu yang berbeda. Objek inilah yang merupakan George Washington dari tahun 1789 hingga 1797, John Adams dari tahun 1797 hingga 1801, dan seterusnya, seperti pada Gambar 3.3. Untuk mengatakan bahwa George Washington adalah presiden sepanjang tahun 1790, kita dapat menulis:

$T(\text{Equals}(\text{President}(\text{USA}), \text{GeorgeWashington}), \text{AD1790})$.

Kita menggunakan simbol fungsi *SamaDengan* (*Equals*) daripada predikat logis standar $=$, karena kita tidak dapat memiliki predikat sebagai argumen untuk T , dan karena interpretasinya bukanlah bahwa *GeorgeWashington* dan *Presiden(USA)* secara logis identik pada tahun 1790; identitas logis bukanlah sesuatu yang dapat berubah seiring waktu. Identitas tersebut ada di antara subkejadian (*subevents*) dari setiap objek yang ditentukan oleh periode tahun 1790.

3.4 KEJADIAN MENTAL DAN OBJEK MENTAL

Agen yang telah kita buat sejauh ini memiliki keyakinan dan dapat menyimpulkan keyakinan baru. Namun, tidak satu pun dari mereka yang memiliki pengetahuan *tentang* keyakinan atau *tentang* deduksi. Pengetahuan tentang pengetahuan dan proses penalaran diri sendiri berguna untuk mengontrol inferensi. Misalnya, misalkan Alice bertanya "berapa akar kuadrat dari 1764" dan Bob menjawab "Saya tidak tahu." Jika Alice bersikeras "berpikirlah lebih keras," Bob harus menyadari bahwa dengan sedikit pemikiran lagi, pertanyaan ini sebenarnya dapat dijawab. Di sisi lain, jika pertanyaannya adalah "Apakah ibumu sedang duduk sekarang?" maka Bob harus menyadari bahwa berpikir lebih keras sepertinya tidak akan membantu. Pengetahuan tentang pengetahuan agen lain juga penting; Bob harus menyadari bahwa ibunya tahu apakah dia sedang duduk atau tidak, dan menanyakannya adalah cara untuk mengetahuinya.

Yang kita butuhkan adalah model objek mental yang ada di kepala seseorang (atau basis pengetahuan sesuatu) dan proses mental yang memanipulasi objek mental tersebut. Modelnya tidak harus detail. Kita tidak perlu bisa memprediksi berapa milidetik yang dibutuhkan agen tertentu untuk membuat deduksi. Kita akan senang hanya bisa menyimpulkan bahwa ibu tahu apakah dia sedang duduk atau tidak.

Kita mulai dengan **sikap proposisional** yang dapat dimiliki agen terhadap objek mental: sikap seperti *Percaya*, *Tahu*, *Menginginkan*, *Berniat*, dan *Menginformasikan*. Kesulitannya adalah sikap-sikap ini tidak berperilaku seperti predikat "normal". Misalnya, misalkan kita mencoba menegaskan bahwa Lois tahu Superman bisa terbang:

Knows(Lois, CanFly(Superman)).

Salah satu masalah kecil di sini adalah bahwa kita biasanya menganggap *CanFly(Superman)* sebagai sebuah kalimat, tetapi di sini ia muncul sebagai sebuah istilah. Masalah itu dapat diatasi hanya dengan mereifikasi *CanFly(Superman)*; menjadikannya sebuah fluent. Masalah yang lebih serius adalah, jika memang benar bahwa Superman adalah Clark Kent, maka kita harus menyimpulkan bahwa Lois tahu bahwa Clark bisa terbang:

$$\begin{aligned} & (Superman = Clark) \wedge Knows(Lois, CanFly(Superman)) \\ & \models Knows(Lois, CanFly(Clark)). \end{aligned}$$

Ini adalah konsekuensi dari fakta bahwa penalaran kesetaraan dibangun ke dalam logika. Biasanya itu adalah hal yang baik; jika agen kita tahu bahwa $2 + 2 = 4$ dan $4 < 5$, maka kita ingin agen kita tahu bahwa $2 + 2 < 5$. Properti ini disebut **transparansi referensial**—tidak peduli istilah apa yang digunakan logika untuk merujuk pada suatu objek, yang penting adalah objek yang dinamakan istilah tersebut. Tetapi untuk sikap proposisional seperti *percaya* (Believe) dan *tahu* (Know), kita ingin memiliki opasitas referensial—istilah yang digunakan itu memang penting, karena tidak semua agen tahu istilah mana yang bersifat koreferensial.

Logika modal dirancang untuk mengatasi masalah ini. Logika biasa hanya berfokus pada satu modalitas, yaitu modalitas kebenaran, yang memungkinkan kita untuk menyatakan "*P* adalah benar." Logika modal mencakup operator modal khusus yang mengambil kalimat (bukan istilah) sebagai argumen. Misalnya, "*A* tahu *P*" direpresentasikan dengan notasi $K_A P$, di mana *K* adalah operator modal untuk pengetahuan. Operator ini membutuhkan dua argumen, yaitu agen (ditulis sebagai subskrip) dan sebuah kalimat. Sintaksis logika modal sama dengan logika tingkat pertama, kecuali bahwa kalimat juga dapat dibentuk dengan operator modal.

Semantik logika modal lebih rumit. Dalam logika tingkat pertama, **model** berisi sekumpulan objek dan interpretasi yang memetakan setiap nama ke objek, relasi, atau fungsi yang sesuai. Dalam logika modal, kita ingin dapat mempertimbangkan kemungkinan bahwa identitas rahasia Superman adalah Clark dan juga kemungkinan bahwa bukan. Oleh karena itu, kita akan membutuhkan model yang lebih rumit, yang terdiri dari kumpulan **dunia yang mungkin** daripada hanya satu dunia nyata. Dunia-dunia tersebut terhubung dalam sebuah grafik oleh **relasi aksesibilitas**, satu relasi untuk setiap operator modal. Kita mengatakan bahwa dunia ω_1 dapat diakses dari dunia ω_0 sehubungan dengan operator modal K_A jika segala sesuatu di ω_1 konsisten dengan apa yang *A* ketahui di ω_0 , dan kita menuliskannya sebagai $Acc(K_A, \omega_0, \omega_1)$. Dalam diagram seperti Gambar 3.4, kita menunjukkan aksesibilitas sebagai panah antara dunia yang mungkin. Sebagai contoh, di dunia nyata, Bukares adalah ibu kota Rumania, tetapi bagi agen yang tidak mengetahui hal itu, dunia-dunia lain yang mungkin dapat diakses, termasuk dunia di mana ibu kota Rumania adalah Sibiu atau Sofia. Mungkin dunia di mana $2+2=5$ tidak akan dapat diakses oleh agen mana pun.

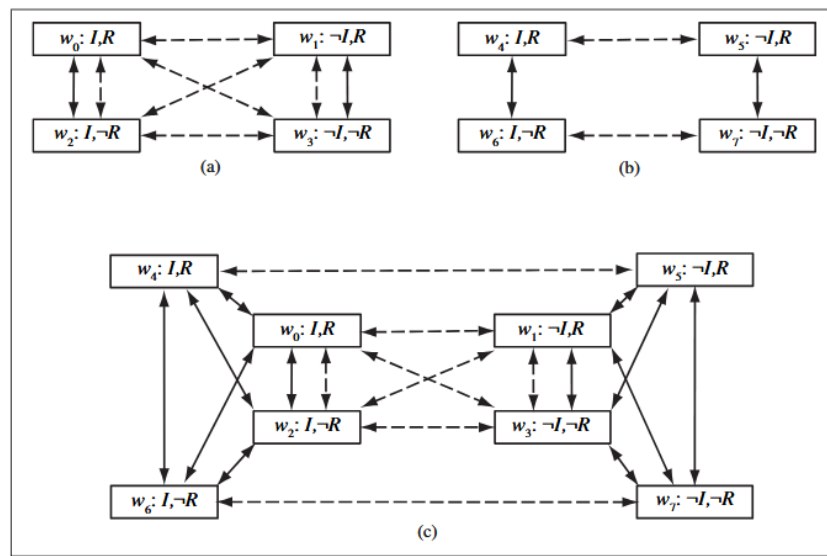
Secara umum, sebuah atom pengetahuan $K_A P$ adalah benar di dunia *w* jika dan hanya jika *P* adalah benar di setiap dunia yang dapat diakses dari *w*. Kebenaran dari kalimat yang lebih kompleks diturunkan dengan aplikasi rekursif dari aturan ini dan aturan normal logika tingkat pertama. Ini berarti logika modal dapat digunakan untuk bernalar tentang kalimat pengetahuan bersarang: apa yang diketahui satu agen tentang pengetahuan agen lain. Misalnya, kita bisa mengatakan bahwa, meskipun Lois tidak tahu apakah identitas rahasia Superman adalah Clark Kent, dia tahu bahwa Clark tahu:

$$K_{Lois} [K_{Clark} Identity(Superman, Clark) \vee K_{Clark} \neg Identity(Superman, Clark)]$$

Gambar 3.4 menunjukkan beberapa kemungkinan dunia untuk domain ini, dengan hubungan aksesibilitas untuk Lois dan Superman.

Dalam diagram KIRI ATAS, adalah pengetahuan umum bahwa Superman mengetahui identitasnya sendiri, dan baik dia maupun Lois belum melihat laporan Weather. Jadi di dunia w_0 dunia w_0 dan w_2 dapat diakses oleh Superman; mungkin hujan diprediksi, mungkin tidak. Bagi Lois, keempat dunia dapat diakses satu sama lain; dia tidak tahu apa-apa tentang laporan Weather atau apakah Clark adalah Superman. Tapi dia tahu bahwa Superman tahu apakah dia Clark, karena di setiap dunia yang dapat diakses oleh Lois, Superman tahu identitasnya I atau dia tahu bukan identitasnya $\neg I$. Lois tidak tahu mana yang benar, tetapi bagaimanapun dia tahu Superman tahu.

Dalam diagram KANAN ATAS, adalah pengetahuan umum bahwa Lois telah melihat laporan Weather. Jadi di dunia w_4 dia tahu bahwa hujan diprediksi, dan di dunia w_6 dia tahu bahwa hujan tidak diprediksi.



Gambar 3.4 menunjukkan Dunia yang mungkin dengan relasi aksesibilitas $K_{Superman}$ (panah padat) dan K_{Lois} (panah putus-putus). Proposisi R berarti "laporan Weather besok adalah hujan" dan I berarti "identitas rahasia Superman adalah Clark Kent." Semua dunia dapat diakses oleh dirinya sendiri; panah dari sebuah dunia ke dirinya sendiri tidak ditampilkan.

Superman tidak mengetahui laporan Weather, tetapi dia tahu bahwa Lois mengetahuinya, karena di setiap dunia yang dapat diakses olehnya, Lois tahu R (hujan diprediksi) atau dia tahu $\neg R$ (hujan tidak diprediksi).

Dalam diagram BAWAH, kita merepresentasikan skenario di mana adalah pengetahuan umum bahwa Superman mengetahui identitasnya, dan Lois mungkin telah melihat atau mungkin belum melihat laporan Weather. Kita merepresentasikan ini dengan menggabungkan dua skenario di atas, dan menambahkan panah untuk menunjukkan bahwa Superman tidak tahu skenario mana yang sebenarnya terjadi. Lois memang tahu, jadi kita tidak perlu menambahkan panah apa pun untuknya. Di dunia w_0 , Superman masih tahu I (identitas rahasianya), tetapi tidak tahu R (laporan Weather). Sekarang dia tidak tahu apakah Lois tahu R . Dari apa yang diketahui Superman, dia mungkin berada di dunia w_0 atau w_2 dalam hal ini Lois

tidak tahu apakah R benar. Atau dia bisa berada di dunia w_4 , dalam hal ini Lois tahu R , atau w_6 , dalam hal ini Lois tahu $\neg R$.

Ada banyak sekali dunia yang mungkin, jadi kuncinya adalah memperkenalkan dunia yang Anda butuhkan untuk merepresentasikan apa yang sedang Anda coba modelkan. Sebuah dunia baru yang mungkin dibutuhkan untuk membicarakan fakta-fakta yang berbeda (misalnya, hujan diprediksi atau tidak), atau untuk membicarakan keadaan pengetahuan yang berbeda (misalnya, apakah Lois tahu bahwa hujan diprediksi). Itu berarti dua dunia yang mungkin, seperti w_4 dan w_0 dalam Gambar 3.4, mungkin memiliki fakta dasar yang sama tentang dunia, tetapi berbeda dalam relasi aksesibilitas mereka, dan oleh karena itu berbeda dalam fakta tentang pengetahuan.

Logika modal memecahkan beberapa masalah rumit terkait interaksi antara pengkuantifikasi dan pengetahuan. Kalimat bahasa Inggris "Bond tahu bahwa seseorang adalah mata-mata" itu ambigu. Penafsiran pertama adalah bahwa ada seseorang tertentu yang Bond tahu adalah mata-mata; kita bisa menuliskannya sebagai:

$$\exists x K_{Bond} Spy(x),$$

yang dalam logika modal berarti ada x yang, di semua dunia yang dapat diakses, Bond tahu bahwa itu adalah mata-mata. Pembacaan kedua adalah bahwa Bond hanya tahu bahwa setidaknya ada satu mata-mata:

$$K_{Bond} \exists x Spy(x).$$

Penafsiran logika modal adalah bahwa di setiap dunia yang dapat diakses terdapat x yang merupakan mata-mata, tetapi x tersebut tidak harus sama di setiap dunia.

Setelah kita memiliki operator modal untuk pengetahuan, kita dapat menulis aksioma untuk itu. Pertama, kita dapat mengatakan bahwa agen mampu menarik deduksi; jika seorang agen tahu P dan tahu bahwa P menyiratkan Q , maka agen tersebut tahu Q :

$$(K_A P \wedge K_A (P \Rightarrow Q)) \Rightarrow K_A Q.$$

Dari sini (dan beberapa aturan lain tentang identitas logis), kita dapat menetapkan bahwa $K_A (P \vee \neg P)$ adalah sebuah tautologi; setiap agen tahu bahwa setiap proposisi P itu benar atau salah. Di sisi lain, $(K_A P) \vee (K_A \neg P)$ bukanlah sebuah tautologi; secara umum, akan ada banyak proposisi yang tidak diketahui kebenarannya oleh agen dan juga tidak diketahui kesalahannya oleh agen.

Dikatakan (kembali ke Plato) bahwa pengetahuan adalah keyakinan benar yang dibenarkan. Artinya, jika itu benar, jika Anda mempercayainya, dan jika Anda memiliki alasan yang tak terbantahkan, maka Anda mengetahuinya. Itu berarti bahwa jika Anda mengetahui sesuatu, itu pasti benar, dan kita memiliki aksioma:

$$K_A P \Rightarrow P$$

Lebih jauh lagi, agen logis harus mampu melakukan introspeksi terhadap pengetahuan mereka sendiri. Jika mereka mengetahui sesuatu, maka mereka tahu bahwa mereka mengetahuinya:

$$K_a P \Rightarrow K_a(K_a P) .$$

Kita dapat mendefinisikan aksioma serupa untuk keyakinan (sering dilambangkan dengan **B**) dan modalitas lainnya. Namun, salah satu masalah dengan pendekatan logika modal adalah ia mengasumsikan **kemahatahuan logis** pada agen. Artinya, jika agen mengetahui sekumpulan aksioma, maka agen tersebut mengetahui semua konsekuensi dari aksioma tersebut. Ini berada di landasan yang goyah bahkan untuk gagasan pengetahuan yang agak abstrak, tetapi tampaknya lebih buruk lagi untuk keyakinan, karena keyakinan memiliki konotasi yang lebih merujuk pada hal-hal yang direpresentasikan secara fisik dalam agen, bukan hanya yang berpotensi dapat diturunkan. Ada upaya untuk mendefinisikan bentuk **rasionalitas terbatas** untuk agen; untuk mengatakan bahwa agen percaya pada pernyataan-pernyataan yang dapat diturunkan dengan aplikasi tidak lebih dari k langkah penalaran, atau tidak lebih dari s detik komputasi. Upaya-upaya ini umumnya tidak memuaskan.

3.5 SISTEM PENALARAN UNTUK KATEGORI

Kategori adalah blok bangunan utama dari skema representasi pengetahuan skala besar. Bagian ini menjelaskan sistem yang dirancang khusus untuk mengatur dan bernalar dengan kategori. Ada dua keluarga sistem yang sangat erat kaitannya: **jaringan semantik** menyediakan bantuan grafis untuk memvisualisasikan basis pengetahuan dan algoritma yang efisien untuk menyimpulkan properti suatu objek berdasarkan keanggotaan kategorinya; dan **logika deskripsi** menyediakan bahasa formal untuk membangun dan menggabungkan definisi kategori serta algoritma yang efisien untuk memutuskan hubungan himpunan bagian dan superset antar kategori.

Jaringan semantik

Pada tahun 1909, Charles S. Peirce mengusulkan notasi grafis berupa simpul dan tepi yang disebut **grafik eksistensial** yang dia sebut sebagai "logika masa depan." Demikianlah dimulai perdebatan panjang antara pendukung "logika" dan pendukung "jaringan semantik." Sayangnya, perdebatan tersebut mengaburkan fakta bahwa jaringan semantik—setidaknya yang memiliki semantik yang terdefinisi dengan baik—adalah bentuk logika. Notasi yang disediakan jaringan semantik untuk jenis kalimat tertentu seringkali lebih nyaman, tetapi jika kita melepaskan masalah "antarmuka manusia," konsep-konsep dasarnya—objek, relasi, kuantifikasi, dan sebagainya—adalah sama.

Ada banyak varian jaringan semantik, tetapi semuanya mampu merepresentasikan objek individual, kategori objek, dan hubungan antar objek. Notasi grafis yang umum menampilkan nama objek atau kategori dalam oval atau kotak, dan menghubungkannya dengan tautan berlabel. Misalnya, Gambar 3.5 memiliki tautan *AnggotaDari* antara *Mary* dan *Wanita*, yang

sesuai dengan pernyataan logis " $Mary \in Wanita$ ". Demikian pula, tautan *SaudariDari* antara *Mary* dan *John* sesuai dengan pernyataan " $SaudariDari(Mary, John)$ ". Kita dapat menghubungkan kategori menggunakan **tautanDari**, dan seterusnya. Sangat menyenangkan menggambar gelembung dan panah sehingga seseorang bisa terbawa suasana. Misalnya, kita tahu bahwa manusia memiliki ibu yang adalah wanita, jadi bisakah kita menggambar tautan *MemilikiIbu* dari *seseorang* ke *Wanita*? Jawabannya adalah tidak, karena *MemilikiIbu* adalah hubungan antara seseorang dengan ibunya, dan kategori tidak memiliki ibu.

Karena alasan ini, kami telah menggunakan notasi khusus—tautan kotak ganda—pada Gambar 3.5. Tautan ini menegaskan bahwa

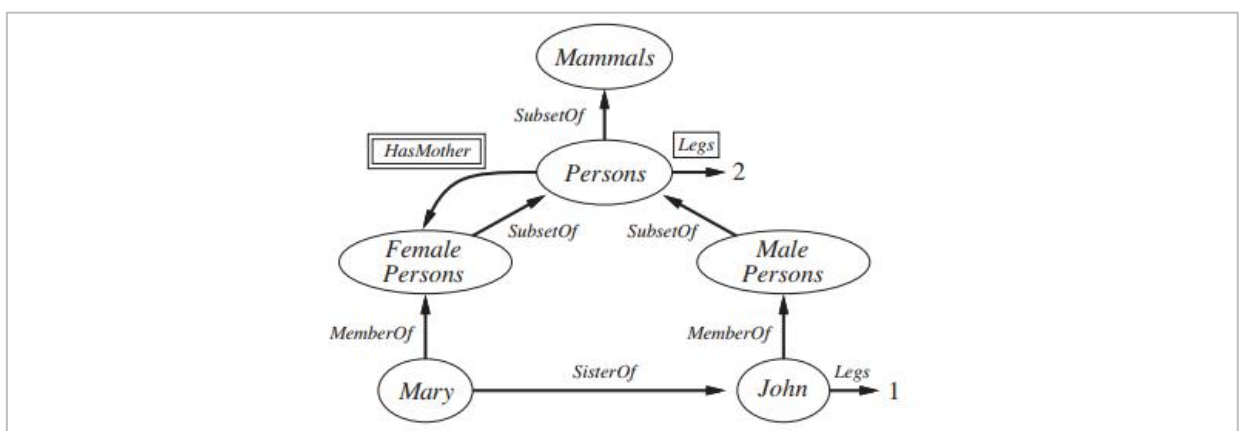
$$\forall x \ x \in Persons \Rightarrow [\forall y \ HasMother(x,y) \Rightarrow y \in FemalePersons].$$

Kita mungkin juga ingin menegaskan bahwa manusia memiliki dua kaki—yaitu,

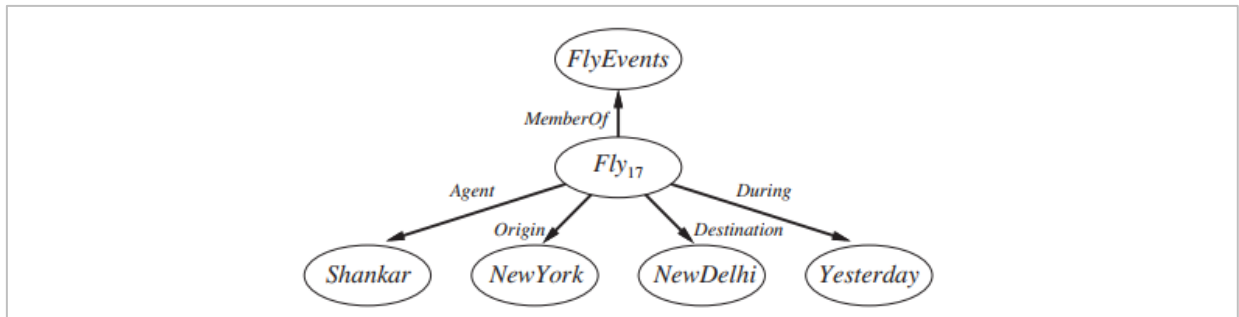
$$\forall x \ x \in Persons \Rightarrow Legs(x, 2).$$

Seperti sebelumnya, kita perlu berhati-hati untuk tidak menegaskan bahwa suatu kategori memiliki kaki; tautan kotak tunggal pada Gambar 3.5 digunakan untuk menegaskan properti setiap anggota suatu kategori.

Notasi jaringan semantik memudahkan untuk melakukan penalaran **pewarisan** seperti yang diperkenalkan pada Bagian selanjutnya. Misalnya, karena *Mary* adalah seorang individu, dia mewarisi properti memiliki dua kaki. Jadi, untuk mengetahui berapa banyak kaki yang dimiliki *Mary*, algoritma pewarisan mengikuti *tautanAnggota* dari *Mary* ke kategori tempat dia berada, lalu mengikuti tautan *HimpunanBagianDari* ke atas hierarki hingga menemukan kategori di mana ada tautan *Kaki* yang berada dalam kotak—dalam kasus ini, kategori *Individu*. Kesederhanaan dan efisiensi mekanisme inferensi ini, dibandingkan dengan pembuktian teorema logis, telah menjadi salah satu daya tarik utama jaringan semantik.



Gambar 3.5 Jaringan semantik dengan empat objek (*John*, *Mary*, 1, dan 2) dan empat kategori. Relasi dilambangkan dengan tautan berlabel.



Gambar 3.6 Sebuah fragmen jaringan semantik yang menunjukkan representasi pernyataan logis Fly(Shankar, NewYork, NewDelhi, Yesterday).

Pewarisan menjadi rumit ketika sebuah objek dapat termasuk dalam lebih dari satu kategori atau ketika sebuah kategori dapat menjadi bagian dari lebih dari satu kategori lain; ini disebut **pewarisan ganda**. Dalam kasus seperti ini, algoritma pewarisan mungkin menemukan dua atau lebih nilai yang bertentangan yang menjawab pertanyaan. Karena alasan ini, pewarisan ganda dilarang dalam beberapa bahasa **pemrograman berorientasi objek** (OOP), seperti Java, yang menggunakan pewarisan dalam hierarki kelas. Biasanya diizinkan dalam jaringan semantik.

Pembaca mungkin telah memperhatikan kelemahan yang jelas dari notasi jaringan semantik, dibandingkan dengan logika tingkat pertama: fakta bahwa tautan antar gelembung hanya merepresentasikan relasi biner. Misalnya, kalimat "*Terbang(Shankar, NewYork, NewDelhi, Yesterday)*" tidak dapat langsung ditegaskan dalam jaringan semantik. Meskipun demikian, kita dapat memperoleh efek dari penegasan n -ary dengan mereifikasi proposisi itu sendiri sebagai sebuah peristiwa yang termasuk dalam kategori peristiwa yang sesuai. Gambar 3.6 menunjukkan struktur jaringan semantik untuk peristiwa khusus ini. Perhatikan bahwa pembatasan pada relasi biner memaksa terciptanya ontologi konsep tereifikasi yang kaya.

Mereifikasi proposisi memungkinkan untuk merepresentasikan setiap kalimat atomik dasar bebas fungsi dari logika tingkat pertama dalam notasi jaringan semantik. Jenis-jenis kalimat berkuantor universal tertentu dapat ditegaskan menggunakan tautan invers dan panah berkotak tunggal serta berkotak ganda yang diterapkan pada kategori, tetapi itu masih jauh dari logika tingkat pertama yang lengkap. Negasi, disjungsi, simbol fungsi bersarang, dan kuantifikasi eksistensial semuanya hilang. Sekarang, adalah *mungkin* untuk memperluas notasi agar setara dengan logika tingkat pertama—seperti dalam grafik eksistensial Peirce—tetapi melakukannya meniadakan salah satu keuntungan utama jaringan semantik, yaitu kesederhanaan dan transparansi proses inferensi. Para perancang dapat membangun jaringan besar dan masih memiliki gambaran yang jelas tentang kueri apa yang akan efisien, karena (a) mudah untuk memvisualisasikan langkah-langkah yang akan dilalui oleh prosedur inferensi dan (b) dalam beberapa kasus bahasa kueri sangat sederhana sehingga kueri yang sulit tidak dapat diajukan. Dalam kasus di mana kekuatan ekspresif terbukti terlalu membatasi, banyak sistem jaringan semantik menyediakan **lampiran prosedural** untuk mengisi celah. Lampiran prosedural adalah teknik di mana kueri tentang (atau terkadang penegasan) relasi tertentu menghasilkan panggilan ke prosedur khusus yang dirancang untuk relasi tersebut, daripada algoritma inferensi umum.

Salah satu aspek terpenting dari jaringan semantik adalah kemampuannya untuk merepresentasikan **nilai default** untuk kategori. Memeriksa Gambar 3.5 dengan cermat, seseorang akan melihat bahwa John memiliki satu kaki, meskipun dia adalah seorang individu dan semua individu memiliki dua kaki. Dalam KB (Basis Pengetahuan) yang strictly logis, ini akan menjadi kontradiksi, tetapi dalam jaringan semantik, pernyataan bahwa semua individu memiliki dua kaki hanya berstatus *default*; artinya, seorang individu diasumsikan memiliki dua kaki kecuali jika ini dibantah oleh informasi yang lebih spesifik. Semantik *default* ini ditegakkan secara alami oleh algoritma pewarisan, karena ia mengikuti tautan ke atas dari objek itu sendiri (John dalam kasus ini) dan berhenti segera setelah menemukan suatu nilai. Kita mengatakan bahwa nilai bawaan **digantikan** oleh nilai yang lebih spesifik. Perhatikan bahwa kita juga bisa menimpa jumlah kaki *default* dengan membuat kategori *IndividuBerkakiSatu*, sebuah himpunan bagian dari *Individu* di mana *John* adalah anggotanya.

Kita dapat mempertahankan semantik yang murni logis untuk jaringan jika kita mengatakan bahwa penegasan *Kaki* untuk *Individu* mencakup pengecualian untuk John:

$$\forall x x \in \text{Persons} \wedge x \neq \text{John} \Rightarrow \text{Legs}(x, 2).$$

Untuk jaringan yang *tetap*, ini secara semantik memadai tetapi akan jauh kurang ringkas dibandingkan notasi jaringan itu sendiri jika ada banyak pengecualian. Namun, untuk jaringan yang akan diperbarui dengan lebih banyak penegasan, pendekatan semacam itu gagal—kita benar-benar ingin mengatakan bahwa setiap individu yang belum diketahui memiliki satu kaki juga merupakan pengecualian. Bagian selanjutnya akan membahas lebih dalam masalah ini dan penalaran default secara umum.

Logika deskripsi

Sintaksis logika tingkat pertama dirancang untuk memudahkan menyatakan hal-hal tentang objek. Sementara itu, **logika deskripsi** adalah notasi yang dirancang untuk memudahkan mendefinisikan dan menggambarkan properti kategori. Sistem logika deskripsi berevolusi dari jaringan semantik sebagai respons terhadap tekanan untuk memformalkan makna jaringan sambil tetap mempertahankan penekanan pada struktur taksonomi sebagai prinsip pengorganisasi.

Tugas inferensi utama untuk logika deskripsi adalah **subsumpsi** (memeriksa apakah satu kategori adalah subset dari kategori lain dengan membandingkan definisinya) dan **klasifikasi** (memeriksa apakah suatu objek termasuk dalam suatu kategori). Beberapa sistem juga mencakup **konsistensi** definisi kategori—apakah kriteria keanggotaannya dapat dipenuhi secara logis.

```

Concept → Thing | ConceptName
          | And(Concept, ...)
          | All(RoleName, Concept)
          | AtLeast(Integer, RoleName)
          | AtMost(Integer, RoleName)

```



```

| Fills(RoleName, IndividualName, ...)
| SameAs(Path, Path)
| OneOf(IndividualName, ...)
Path → [RoleName, ...]

```

Gambar 3.7 Sintaksis deskripsi dalam subset bahasa CLASSIC.

Bahasa CLASSIC adalah logika deskripsi yang tipikal. Sintaksis deskripsi CLASSIC ditunjukkan pada Gambar 3.7. Misalnya, untuk mengatakan bahwa bujangan adalah pria dewasa yang belum menikah, kita akan menulis:

Bachelor = And(Unmarried, Adult, Male) .

Padanannya dalam logika orde pertama adalah

Bachelor(x) ⇔ Unmarried(x) ∧ Adult(x) ∧ Male(x) .

Perhatikan bahwa logika deskripsi memiliki aljabar operasi pada predikat, yang tentu saja tidak bisa kita lakukan dalam logika tingkat pertama. Setiap deskripsi dalam CLASSIC dapat diterjemahkan ke dalam kalimat tingkat pertama yang setara, tetapi beberapa deskripsi lebih lugas dalam CLASSIC. Misalnya, untuk menggambarkan kumpulan pria dengan setidaknya tiga putra yang semuanya menganggur dan menikah dengan dokter, serta paling banyak dua putri yang semuanya adalah profesor di departemen fisika atau matematika, kita akan menggunakan:

*And(Man, AtLeast(3, Son), AtMost(2, Daughter),
 All(Son, And(Unemployed, Married, All(Spouse, Doctor))),
 All(Daughter , And(Professor , Fills(Department, Physics, Math)))) .*

Kita biarkan ini sebagai latihan untuk menerjemahkannya ke dalam logika tingkat pertama.

Mungkin aspek terpenting dari logika deskripsi adalah penekanannya pada kemudahan inferensi. Sebuah contoh masalah diselesaikan dengan mendeskripsikannya dan kemudian menanyakan apakah ia tercakup oleh salah satu dari beberapa kategori solusi yang mungkin. Dalam sistem logika tingkat pertama standar, memprediksi waktu solusi seringkali tidak mungkin. Seringkali pengguna harus merekrut representasi untuk menghindari kumpulan kalimat yang tampaknya menyebabkan sistem membutuhkan waktu beberapa minggu untuk memecahkan masalah. Dorongan dalam logika deskripsi, di sisi lain, adalah untuk memastikan bahwa pengujian subsumpsi dapat diselesaikan dalam waktu polinomial dalam ukuran deskripsi.⁷

Hal ini terdengar luar biasa secara prinsip, sampai seseorang menyadari bahwa ini hanya dapat memiliki salah satu dari dua konsekuensi: entah masalah sulit tidak dapat dinyatakan sama sekali, atau mereka membutuhkan deskripsi yang secara eksponensial besar! Namun, hasil

kemudahan ini memang menjelaskan jenis konstruksi apa yang menyebabkan masalah dan dengan demikian membantu pengguna untuk memahami bagaimana representasi yang berbeda berperilaku. Misalnya, logika deskripsi biasanya tidak memiliki *negasi* dan *disjungsi*. Masing-masing memaksa sistem logis tingkat pertama untuk melalui analisis kasus yang berpotensi eksponensial untuk memastikan kelengkapan. CLASSIC hanya mengizinkan bentuk disjungsi yang terbatas dalam konstruksi *Mengisi* dan *SalahSatu*, yang mengizinkan disjungsi atas individu-individu yang disebutkan secara eksplisit tetapi tidak atas deskripsi. Dengan deskripsi disjuntif, definisi bersarang dapat dengan mudah menyebabkan jumlah rute alternatif yang eksponensial di mana satu kategori dapat mencakup kategori lainnya.

3.6 PENALARAN DENGAN INFORMASI DEFAULT

Pada bagian sebelumnya, kita melihat contoh sederhana dari penegasan dengan status standar: manusia memiliki dua kaki. Standar ini dapat ditimpa oleh informasi yang lebih spesifik, seperti Long John Silver yang memiliki satu kaki. Kita melihat bahwa mekanisme pewarisan dalam jaringan semantik mengimplementasikan penimpaan standar dengan cara yang sederhana dan alami. Dalam bagian ini, kita mempelajari standar secara lebih umum, dengan tujuan memahami semantik standar daripada hanya menyediakan mekanisme prosedural.

Pembatasan dan logika default

Dalam bab ini, kita melihat bahwa properti yang diwarisi oleh semua anggota suatu kategori dalam jaringan semantik dapat ditimpa oleh informasi yang lebih spesifik untuk subkategori. Kita melihat bahwa di bawah **asumsi dunia tertutup**, jika proposisi α tidak disebutkan dalam KB , maka $KB \models \neg\alpha$, tetapi $KB \wedge \alpha \models \alpha$.

Introspeksi sederhana menunjukkan bahwa kegagalan monotonisitas ini meluas dalam penalaran akal sehat. Tampaknya manusia sering "buru-buru menyimpulkan." Misalnya, ketika seseorang melihat mobil diparkir di jalan, seseorang biasanya bersedia percaya bahwa mobil itu memiliki empat roda meskipun hanya tiga yang terlihat. Nah, teori probabilitas tentu dapat memberikan kesimpulan bahwa roda keempat ada dengan probabilitas tinggi, namun, bagi kebanyakan orang, kemungkinan mobil tidak memiliki empat roda *tidak muncul kecuali ada bukti baru yang muncul*. Jadi, tampaknya kesimpulan empat roda dicapai secara *default*, tanpa alasan untuk meragukannya. Jika bukti baru tiba—misalnya, jika seseorang melihat pemilik membawa roda dan melihat mobil di dongkrak—maka kesimpulan dapat ditarik kembali. Jenis penalaran ini dikatakan menunjukkan **non-monotonisitas**, karena kumpulan keyakinan tidak tumbuh secara monoton seiring waktu saat bukti baru tiba. **Logika non-monotonik** telah dirancang dengan pengertian kebenaran dan implikasi yang dimodifikasi untuk menangkap perilaku tersebut. Kita akan melihat dua logika semacam itu yang telah dipelajari secara ekstensif: pembatasan dan logika default.

Pembatasan dapat dipandang sebagai versi yang lebih kuat dan tepat dari asumsi dunia tertutup. Idennya adalah untuk menentukan predikat-predikat tertentu yang diasumsikan "palsu sebanyak mungkin"—yaitu, palsu untuk setiap objek kecuali objek-objek yang diketahui kebenarannya. Misalnya, misalkan kita ingin menegaskan aturan default bahwa burung terbang. Kita akan memperkenalkan sebuah predikat, katakanlah *Abnormal* $1(x)$, dan menulis:

$$Bird(x) \wedge \neg Abnormal_1(x) \Rightarrow Flies(x).$$

Jika kita mengatakan bahwa $Abnormal_1$ harus **dipembatasan**, penalaran sirkumskriptif berhak mengasumsikan $\neg Abnormal_1(x)$ kecuali jika $\neg Abnormal_1(x)$ diketahui benar. Ini memungkinkan kesimpulan $Terbang(Tweety)$ ditarik dari premis $Burung(Tweety)$, tetapi kesimpulan tersebut tidak lagi berlaku jika $Abnormal_1(Tweety)$ ditegaskan.

Pembatasan dapat dipandang sebagai contoh logika **preferensi model**. Dalam logika semacam itu, suatu kalimat dianggap logis (dengan status default) jika kalimat itu benar *di semua* model KB yang lebih disukai, berbeda dengan persyaratan kebenaran di semua model dalam logika klasik. Untuk pembatasan, satu model lebih disukai daripada yang lain jika model tersebut memiliki lebih sedikit objek abnormal. Mari kita lihat bagaimana ide ini bekerja dalam konteks pewarisan ganda dalam jaringan semantik. Contoh standar di mana pewarisan ganda bermasalah disebut "berlian Nixon." Ini muncul dari pengamatan bahwa Richard Nixon adalah seorang Quaker (dan oleh karena itu secara *default* seorang pasifis) dan seorang Republikan (dan oleh karena itu secara *default* bukan seorang pasifis). Kita bisa menuliskannya sebagai berikut:

$$\begin{aligned} &Republican(Nixon) \wedge Quaker(Nixon). \\ &Republican(x) \wedge \neg Abnormal_2(x) \Rightarrow \neg Pacifist(x). \\ &Quaker(x) \wedge \neg Abnormal_3(x) \Rightarrow Pacifist(x). \end{aligned}$$

Jika kita melakukan pembatasan terhadap $Abnormal_2$ dan $Abnormal_3$, ada dua model yang lebih disukai: satu di mana $Abnormal_2(Nixon)$ dan $Pasifis(Nixon)$ berlaku, dan satu di mana $Abnormal_3(Nixon)$ dan $\neg Pasifis(Nixon)$ berlaku. Dengan demikian, penalaran sirkumskriptif tetap bersikap agnostik secara tepat mengenai apakah Nixon seorang pasifis. Jika kita ingin, sebagai tambahan, menegaskan bahwa keyakinan agama lebih diutamakan daripada keyakinan politik, kita dapat menggunakan formalisme yang disebut **pembatasan terprioritaskan** untuk memberikan preferensi pada model di mana $Abnormal_3$ diminimalkan. Logika default adalah formalisme di mana aturan default dapat ditulis untuk menghasilkan kesimpulan yang kontingen dan tidak monoton. Aturan default terlihat seperti ini:

$$Bird(x) : Flies(x)/Flies(x).$$

Aturan ini berarti bahwa jika $Bird(x)$ benar, dan jika $Flies(x)$ konsisten dengan basis pengetahuan, maka $Flies(x)$ dapat disimpulkan secara default. Secara umum, aturan default memiliki bentuk

$$P : J_1, \dots, J_n / C$$

di mana P disebut prasyarat, C adalah kesimpulan, dan J_i adalah justifikasi—jika salah satu dari mereka dapat dibuktikan salah, maka kesimpulan tidak dapat ditarik. Setiap variabel yang

muncul di J_i atau C juga harus muncul di P . Contoh berlian Nixon dapat direpresentasikan dalam logika *default* dengan satu fakta dan dua aturan *default*:

$Republican(Nixon) \wedge Quaker(Nixon) .$
 $Republican(x) : \neg Pacifist(x) / \neg Pacifist(x) .$
 $Quaker(x) : Pacifist(x) / Pacifist(x) .$

Untuk menafsirkan arti aturan *default*, kita mendefinisikan gagasan **ekstensi** teori default sebagai himpunan konsekuensi maksimal dari teori tersebut. Artinya, sebuah ekstensi S terdiri dari fakta-fakta asli yang diketahui dan sekumpulan kesimpulan dari aturan *default*, sedemikian rupa sehingga tidak ada kesimpulan tambahan yang dapat ditarik dari S dan justifikasi dari setiap kesimpulan *default* di S konsisten dengan S . Seperti halnya model yang lebih disukai dalam pembatasan, kita memiliki dua kemungkinan ekstensi untuk "berlian Nixon": satu di mana dia adalah seorang pasifis dan satu di mana dia bukan. Skema yang diprioritaskan ada di mana beberapa aturan default dapat diberi prioritas di atas yang lain, memungkinkan beberapa ambiguitas untuk diselesaikan.

Sejak tahun 1980, ketika logika non-monoton pertama kali diusulkan, banyak kemajuan telah dicapai dalam memahami sifat-sifat matematisnya. Namun, masih ada pertanyaan-pertanyaan yang belum terpecahkan. Misalnya, jika "Mobil memiliki empat roda" adalah salah, apa artinya memiliki informasi itu dalam basis pengetahuan seseorang? Apa kumpulan aturan default yang baik untuk dimiliki? Jika kita tidak dapat memutuskan, untuk setiap aturan secara terpisah, apakah itu termasuk dalam basis pengetahuan kita, maka kita memiliki masalah serius non-modularitas. Akhirnya, bagaimana keyakinan yang memiliki status default dapat digunakan untuk membuat keputusan? Ini mungkin masalah tersulit untuk penalaran default. Keputusan sering kali melibatkan pertukaran, dan seseorang karenanya perlu membandingkan *kekuatan* keyakinan pada hasil tindakan yang berbeda, dan *biaya* membuat keputusan yang salah. Dalam kasus di mana jenis keputusan yang sama dibuat berulang kali, dimungkinkan untuk menafsirkan aturan default sebagai pernyataan "probabilitas ambang batas". Misalnya, aturan default "Rem saya selalu OK" sebenarnya berarti "Probabilitas bahwa rem saya OK, mengingat tidak ada informasi lain, cukup tinggi sehingga keputusan optimal bagi saya adalah mengemudi tanpa memeriksanya." Ketika konteks keputusan berubah—misalnya, ketika seseorang mengendarai truk yang sangat berat menuruni jalan pegunungan yang curam—aturan default tiba-tiba menjadi tidak tepat, meskipun tidak ada bukti baru tentang rem yang rusak. Pertimbangan-pertimbangan ini telah menyebabkan beberapa peneliti untuk mempertimbangkan bagaimana menyematkan penalaran default dalam teori probabilitas atau teori utilitas.

Sistem pemeliharaan kebenaran

Kami telah melihat bahwa banyak kesimpulan yang ditarik oleh sistem representasi pengetahuan hanya akan memiliki status default, daripada benar-benar pasti. Tak pelak, beberapa dari fakta-fakta yang disimpulkan ini akan ternyata salah dan harus ditarik kembali di hadapan informasi baru. Proses ini disebut **revisi kepercayaan**. Misalkan basis pengetahuan KB berisi kalimat P —mungkin kesimpulan default yang dicatat oleh algoritma perantaraan maju,

atau mungkin hanya sebuah pernyataan yang salah—dan kita ingin menjalankan $TELL(KB, \neg P)$. Untuk menghindari penciptaan kontradiksi, kita harus terlebih dahulu menjalankan $RETRACT(KB, P)$. Ini terdengar cukup mudah. Namun, masalah muncul jika ada kalimat *tambahan* yang disimpulkan dari P dan ditegaskan dalam KB. Misalnya, implikasi $P \Rightarrow Q$ mungkin telah digunakan untuk menambahkan Q . "Solusi" yang jelas—menarik kembali semua kalimat yang disimpulkan dari P —gagal karena kalimat-kalimat tersebut mungkin memiliki pembenaran lain selain P . Misalnya, jika R dan $R \Rightarrow Q$ juga ada di KB , maka Q tidak perlu dihapus sama sekali. *Sistem pemeliharaan kebenaran*, atau TMS, dirancang untuk menangani komplikasi semacam ini.

Salah satu pendekatan sederhana untuk pemeliharaan kebenaran adalah dengan melacak urutan kalimat yang diberitahukan ke basis pengetahuan dengan menomorinya dari P_i hingga P_n . Ketika panggilan $RETRACT(KB, P_i)$ dibuat, sistem kembali ke keadaan tepat sebelum P_i ditambahkan, sehingga menghapus P_i dan semua kesimpulan yang berasal dari P_i . Kalimat P_{i+1} hingga P_n kemudian dapat ditambahkan kembali. Ini sederhana, dan menjamin bahwa basis pengetahuan akan konsisten, tetapi menarik kembali P_i membutuhkan penarikan kembali dan penegasan ulang $n - i$ kalimat serta pembatalan dan pengulangan semua kesimpulan yang ditarik dari kalimat-kalimat tersebut. Untuk sistem di mana banyak fakta ditambahkan—seperti basis data komersial besar—ini tidak praktis.

Pendekatan yang lebih efisien adalah sistem pemeliharaan kebenaran berbasis justifikasi, atau **JTMS**. Dalam JTMS, setiap kalimat dalam basis pengetahuan diberi anotasi dengan **justifikasi** yang terdiri dari kumpulan kalimat tempat kalimat tersebut disimpulkan. Misalnya, jika basis pengetahuan sudah berisi $P \Rightarrow Q$, maka $TELL(P)$ akan menyebabkan Q ditambahkan dengan justifikasi $\{P, P \Rightarrow Q\}$. Secara umum, sebuah kalimat dapat memiliki sejumlah justifikasi. Justifikasi membuat penarikan kembali (retraction) menjadi efisien. Mengingat panggilan $RETRACT(P)$, **JTMS** akan menghapus secara tepat kalimat-kalimat yang mana P adalah anggota dari setiap justifikasinya. Jadi, jika kalimat Q memiliki justifikasi tunggal $\{P, P \Rightarrow Q\}$, kalimat itu akan dihapus; jika ia memiliki justifikasi tambahan $\{P, P \vee R \Rightarrow Q\}$, kalimat itu masih akan dihapus; tetapi jika ia juga memiliki justifikasi $\{R, P \vee R \Rightarrow Q\}$, maka kalimat itu akan dipertahankan. Dengan cara ini, waktu yang dibutuhkan untuk penarikan kembali P hanya bergantung pada jumlah kalimat yang berasal dari P daripada jumlah kalimat lain yang ditambahkan sejak P masuk ke basis pengetahuan.

JTMS mengasumsikan bahwa kalimat yang pernah dipertimbangkan kemungkinan besar akan dipertimbangkan lagi. Oleh karena itu, daripada menghapus kalimat sepenuhnya dari basis pengetahuan ketika kehilangan semua justifikasinya, kita hanya menandai kalimat tersebut sebagai "*keluar*" dari basis pengetahuan. Jika pernyataan berikutnya mengembalikan salah satu justifikasi, maka kita menandai kalimat tersebut sebagai "*masuk*" kembali. Dengan cara ini, JTMS mempertahankan semua rantai inferensi yang digunakannya dan tidak perlu menurunkan kembali kalimat ketika justifikasi menjadi valid lagi.

Selain menangani penarikan informasi yang salah, TMSs dapat digunakan untuk mempercepat analisis beberapa situasi hipotetis. Misalkan, Komite Olimpiade Rumania sedang memilih lokasi untuk acara renang, atletik, dan berkuda di Olimpiade 2048 yang akan diadakan

di Rumania. Misalnya, hipotesis pertama adalah *Site(Swimming, Pitesti)*, *Site(Athletics, Bucharest)*, dan *Site(Equestrian, Arad)*. Banyak penalaran harus dilakukan untuk mengetahui konsekuensi logistik dan, oleh karena itu, keinginan dari pemilihan ini. Jika kita ingin mempertimbangkan *Site(Athletics, Sibiu)* sebagai gantinya, TMS menghindari kebutuhan untuk memulai dari awal. Sebaliknya, kita cukup menarik kembali *Site(Athletics, Bucharest)* dan menegaskan *Site(Athletics, Sibiu)*, dan TMS akan menangani revisi yang diperlukan. Rantai inferensi yang dihasilkan dari pilihan Bucharest dapat digunakan kembali dengan Sibiu, asalkan kesimpulannya sama.

Sebuah sistem pemeliharaan kebenaran berbasis asumsi, atau ATMS, membuat jenis penggantian konteks antara dunia hipotetis ini menjadi sangat efisien. Dalam JTMS, pemeliharaan justifikasi memungkinkan Anda untuk bergerak cepat dari satu keadaan ke keadaan lain dengan membuat beberapa penarikan dan penegasan, tetapi pada waktu tertentu hanya satu keadaan yang terwakili. ATMS merepresentasikan semua keadaan yang pernah dipertimbangkan secara bersamaan. Sementara JTMS hanya memberi label setiap kalimat sebagai "masuk" atau "keluar", ATMS melacak, untuk setiap kalimat, asumsi mana yang akan menyebabkan kalimat tersebut menjadi benar. Dengan kata lain, setiap kalimat memiliki label yang terdiri dari sekumpulan set asumsi. Kalimat tersebut hanya berlaku dalam kasus-kasus di mana semua asumsi dalam salah satu set asumsi berlaku.

Sistem pemeliharaan kebenaran juga menyediakan mekanisme untuk menghasilkan penjelasan. Secara teknis, penjelasan dari sebuah kalimat P adalah sekumpulan kalimat E sedemikian rupa sehingga E menyiratkan P . Jika kalimat-kalimat dalam E sudah diketahui benar, maka E hanya menyediakan dasar yang cukup untuk membuktikan bahwa P pasti terjadi. Tetapi penjelasan juga dapat mencakup asumsi—kalimat yang tidak diketahui benar, tetapi akan cukup untuk membuktikan P jika itu benar. Misalnya, seseorang mungkin tidak memiliki informasi yang cukup untuk membuktikan bahwa mobilnya tidak akan menyala, tetapi penjelasan yang masuk akal mungkin termasuk asumsi bahwa baterainya mati. Ini, dikombinasikan dengan pengetahuan tentang bagaimana mobil beroperasi, menjelaskan perilaku yang tidak teramati. Dalam kebanyakan kasus, kita akan lebih memilih penjelasan E yang minimal, artinya tidak ada subset yang tepat dari E yang juga merupakan penjelasan. ATMS dapat menghasilkan penjelasan untuk masalah "mobil tidak akan menyala" dengan membuat asumsi (seperti "bensin di mobil" atau "baterai mati") dalam urutan apa pun yang kita inginkan, bahkan jika beberapa asumsi bertentangan. Kemudian kita melihat label untuk kalimat "mobil tidak akan menyala" untuk membaca set asumsi yang akan membenarkan kalimat tersebut.

Algoritma persis yang digunakan untuk mengimplementasikan sistem pemeliharaan kebenaran sedikit rumit, dan kami tidak membahasnya di sini. Kompleksitas komputasi masalah pemeliharaan kebenaran setidaknya sama besarnya dengan inferensi proposisional—yaitu, NP-hard. Oleh karena itu, Anda tidak boleh berharap pemeliharaan kebenaran menjadi obat mujarab. Namun, jika digunakan dengan hati-hati, TMS dapat memberikan peningkatan substansial dalam kemampuan sistem logis untuk menangani lingkungan dan hipotesis yang kompleks.

3.7 DUNIA BELANJA INTERNET

Di bagian akhir ini, kita akan menggabungkan semua yang telah kita pelajari untuk menyandikan pengetahuan bagi agen penelitian belanja yang membantu pembeli menemukan penawaran produk di Internet. Agen belanja diberi deskripsi produk oleh pembeli dan memiliki tugas untuk menghasilkan daftar halaman Web yang menawarkan produk tersebut untuk dijual, dan memberi peringkat penawaran mana yang terbaik. Dalam beberapa kasus, deskripsi produk pembeli akan tepat, seperti pada kamera digital Canon Rebel XT_i, dan tugasnya kemudian adalah menemukan toko dengan penawaran terbaik. Dalam kasus lain, deskripsi hanya akan ditentukan sebagian, seperti pada kamera digital di bawah Rp 3.000.000, dan agen harus membandingkan produk yang berbeda.

Example Online Store

Select from our fine line of products:

- [Computers](#)
- [Cameras](#)
- [Books](#)
- [Videos](#)
- [Music](#)

```
<h1>Example Online Store</h1>
<i>Select</i> from our fine line of products:
<ul>
<li> <a href="http://example.com/compu">Computers</a>
<li> <a href="http://example.com/camer">Cameras</a>
<li> <a href="http://example.com/books">Books</a>
<li> <a href="http://example.com/video">Videos</a>
<li> <a href="http://example.com/music">Music</a>
</ul>
```

Gambar 3.8 Halaman Web dari toko daring generik dalam bentuk yang dilihat oleh pengguna manusia dari sebuah browser (atas), dan string HTML yang sesuai seperti yang dilihat oleh browser atau agen belanja (bawah). Dalam HTML, karakter antara < dan > adalah perintah markup yang menentukan bagaimana halaman ditampilkan. Misalnya, string <i>Select</i> berarti beralih ke fon miring, menampilkan kata Select, lalu mengakhiri penggunaan fon miring. Pengidentifikasi halaman seperti `http://example.com/books` disebut *uniform resource locator* (URL). Markup Books berarti membuat hyperlink ke url dengan teks jangkar Books.

Lingkungan agen belanja adalah seluruh World Wide Web dalam segala kompleksitasnya—bukan lingkungan simulasi mainan. Persepsi agen adalah halaman Web, tetapi sementara pengguna Web manusia akan melihat halaman ditampilkan sebagai deretan piksel di layar, agen belanja akan mempersepsikan halaman sebagai string karakter yang terdiri dari kata-kata biasa

yang diselingi dengan perintah pemformatan dalam bahasa markup HTML. Gambar 3.8 menunjukkan halaman Web dan string karakter HTML yang sesuai. Masalah persepsi untuk agen belanja melibatkan ekstraksi informasi yang berguna dari persepsi semacam ini.

Jelas, persepsi pada halaman Web lebih mudah daripada, katakanlah, persepsi saat mengemudi taksi di Kairo. Meskipun demikian, ada komplikasi dalam tugas persepsi Internet. Halaman Web pada Gambar 3.8 sederhana dibandingkan dengan situs belanja sungguhan, yang mungkin menyertakan CSS, cookies, Java, Javascript, Flash, protokol pengecualian robot, HTML yang salah bentuk, file suara, film, dan teks yang hanya muncul sebagai bagian dari gambar JPEG. Agen yang dapat menangani semua Internet hampir sama kompleksnya dengan robot yang dapat bergerak di dunia nyata. Kami akan berkonsentrasi pada agen sederhana yang mengabaikan sebagian besar komplikasi ini.

Tugas pertama agen adalah mengumpulkan penawaran produk yang relevan dengan sebuah query. Jika query-nya adalah "laptop", maka halaman Web dengan ulasan laptop kelas atas terbaru akan relevan, tetapi jika tidak menyediakan cara untuk membeli, itu bukan penawaran. Untuk saat ini, kita dapat mengatakan sebuah halaman adalah penawaran jika berisi kata-kata "beli" atau "harga" atau "tambah ke keranjang" dalam tautan atau formulir HTML di halaman tersebut. Misalnya, jika halaman berisi string dengan format "`<a ...add to cart ...</a>`", maka itu adalah penawaran. Ini bisa direpresentasikan dalam logika orde pertama, tetapi lebih mudah untuk mengodekannya ke dalam kode program.

Mengikuti Tautan

Strateginya adalah memulai dari beranda toko daring dan mempertimbangkan semua halaman yang dapat diakses dengan mengikuti tautan yang relevan. Agen akan memiliki pengetahuan tentang sejumlah toko, misalnya:

Amazon $\in$ *OnlineStores* $\wedge$ *Homepage*(*Amazon*, "amazon.com").

Ebay $\in$ *OnlineStores* $\wedge$ *Homepage*(*Ebay*, "ebay.com").

ExampleStore $\in$ *OnlineStores* $\wedge$ *Homepage*(*ExampleStore*, "example.com").

Toko-toko ini mengklasifikasikan barang-barang mereka ke dalam kategori produk, dan menyediakan tautan ke kategori utama dari beranda mereka. Kategori minor dapat dijangkau melalui serangkaian tautan yang relevan, dan pada akhirnya kita akan mencapai penawaran. Dengan kata lain, suatu halaman relevan dengan kueri jika dapat dijangkau oleh serangkaian tautan kategori yang relevan dari beranda toko, dan kemudian dari satu tautan lagi ke penawaran produk. Kita dapat mendefinisikan relevansi:

Relevant(*page*, *query*) $\Leftrightarrow$

$\exists$ *store*, *home* *store* $\in$ *OnlineStores* $\wedge$ *Homepage*(*store*, *home*)

$\wedge \exists$ *url*, *url*₂ *RelevantChain*(*home*, *url*₂, *query*) $\wedge$ *Link*(*url*₂, *url*)

$\wedge$ *page* = *Contents*(*url*).

Di sini, predikat $Link(from, to)$ berarti ada hyperlink dari URL "from" ke URL "to". Untuk mendefinisikan apa yang terhitung sebagai *RelevantChain*, kita perlu mengikuti bukan sembarang hyperlink, tetapi hanya tautan yang teks anchor terkaitnya menunjukkan bahwa tautan tersebut relevan dengan query produk. Untuk ini, kita menggunakan $LinkText(from, to, text)$ yang berarti ada tautan antara "from" dan "to" dengan "text" sebagai teks anchor-nya. Rantai tautan antara dua URL, start dan end, relevan dengan deskripsi d jika teks anchor dari setiap tautan adalah nama kategori yang relevan untuk d . Keberadaan rantai itu sendiri ditentukan oleh definisi rekursif, dengan rantai kosong ($start = end$) sebagai kasus dasarnya:

$$RelevantChain(start, end, query) \Leftrightarrow (start = end) \vee (\exists u, text \, LinkText(start, u, text) \wedge RelevantCategoryName(query, text) \wedge RelevantChain(u, end, query)) .$$

Sekarang kita harus mendefinisikan apa artinya teks menjadi *RelevantCategoryName* untuk query. Pertama, kita perlu menghubungkan string ke kategori yang dinamainya. Ini dilakukan dengan menggunakan predikat $Name(s, c)$, yang menyatakan bahwa string s adalah nama untuk kategori c —misalnya, kita dapat menyatakan bahwa $Name("laptops", LaptopComputers)$. Beberapa contoh lain dari predikat $Name$ muncul pada Gambar 3.9(b). Berikutnya, kita mendefinisikan relevansi. Misalkan query tersebut adalah "laptops." Maka $RelevantCategoryName(query, text)$ bernilai benar jika salah satu dari berikut ini berlaku:

- Teks dan kueri menamai kategori yang sama—misalnya, "notebooks" dan "laptops."
- Teks tersebut menamai superkategori seperti "komputer."
- Teks tersebut menamai subkategori seperti "notebook ultralight."

$Books \subset Products$ $MusicRecordings \subset Products$ $MusicCDs \subset MusicRecordings$ $Electronics \subset Products$ $DigitalCameras \subset Electronics$ $StereoEquipment \subset Electronics$ $Computers \subset Electronics$ $DesktopComputers \subset Computers$ $LaptopComputers \subset Computers$...	$Name("books", Books)$ $Name("music", MusicRecordings)$ $Name("CDs", MusicCDs)$ $Name("electronics", Electronics)$ $Name("digital cameras", DigitalCameras)$ $Name("stereos", StereoEquipment)$ $Name("computers", Computers)$ $Name("desktops", DesktopComputers)$ $Name("laptops", LaptopComputers)$ $Name("notebooks", LaptopComputers)$...
(a)	(b)

Gambar 3.9 (a) Taksonomi kategori produk. (b) Nama untuk kategori tersebut.

Definisi logis dari *RelevantCategoryName* adalah sebagai berikut:

$$RelevantCategoryName(query, text) \Leftrightarrow \exists c_1, c_2 \, Name(query, c_1) \wedge Name(text, c_2) \wedge (c_1 \subseteq c_2 \vee c_2 \subseteq c_1) .$$

Jika tidak, teks jangkar tidak relevan karena menamai kategori di luar baris ini, seperti "pakaian" atau "halaman & taman".

Untuk mengikuti tautan yang relevan, maka penting untuk memiliki hierarki kategori produk yang kaya. Bagian atas hierarki ini mungkin terlihat seperti Gambar 3.9(a). Tidak akan mungkin untuk mencantumkan semua kategori belanja yang mungkin, karena pembeli selalu dapat memunculkan beberapa keinginan baru dan produsen akan selalu mengeluarkan produk baru untuk memuaskan mereka (penghangat lutut elektrik?). Meskipun demikian, ontologi sekitar seribu kategori akan berfungsi sebagai alat yang sangat berguna bagi sebagian besar pembeli.

Selain hierarki produk itu sendiri, kita juga perlu memiliki kosakata nama yang kaya untuk kategori. Hidup akan jauh lebih mudah jika ada korespondensi satu-satu antara kategori dan rangkaian karakter yang menamainya. Kita telah melihat masalah sinonim—dua nama untuk kategori yang sama, seperti "komputer laptop" dan "laptop." Ada juga masalah ambiguitas—satu nama untuk dua atau lebih kategori yang berbeda. Misalnya, jika kita menambahkan kalimat

Name("CDs", CertificatesOfDeposit)

ke basis pengetahuan pada Gambar 3.9(b), maka "CD" akan menyebutkan dua kategori yang berbeda.

Sinonim dan ambiguitas dapat menyebabkan peningkatan signifikan dalam jumlah jalur yang harus diikuti agen, dan terkadang dapat menyulitkan untuk menentukan apakah halaman tertentu memang relevan. Masalah yang jauh lebih serius adalah rentang deskripsi yang sangat luas yang dapat diketik pengguna dan nama kategori yang dapat digunakan toko. Misalnya, tautan mungkin mengatakan "laptop" saat basis pengetahuan hanya memiliki "laptop" atau pengguna mungkin meminta "komputer yang dapat saya taruh di meja baki kursi pesawat kelas ekonomi." Tidak mungkin untuk menghitung terlebih dahulu semua cara untuk memberi nama kategori, jadi agen harus dapat melakukan penalaran tambahan dalam beberapa kasus untuk menentukan apakah relasi *Name* berlaku. Dalam kasus terburuk, hal ini memerlukan pemahaman bahasa alami yang lengkap. Dalam praktiknya, beberapa aturan sederhana—seperti mengizinkan "laptop" untuk mencocokkan kategori bernama "laptop"—sangatlah membantu.

Dengan definisi logis dari paragraf sebelumnya dan basis pengetahuan yang sesuai tentang kategori produk dan konvensi penamaan, apakah kita siap untuk menerapkan algoritme inferensi guna memperoleh serangkaian penawaran yang relevan untuk kueri kita? Belum juga! Elemen yang hilang adalah fungsi *Contents(url)*, yang merujuk ke halaman HTML di URL tertentu. Agen tidak memiliki konten halaman dari setiap URL dalam basis pengetahuannya; juga tidak memiliki aturan eksplisit untuk menyimpulkan konten tersebut. Sebaliknya, kita dapat mengatur prosedur HTTP yang tepat untuk dijalankan setiap kali subtujuan melibatkan fungsi *Contents*. Dengan cara ini, mesin inferensi akan tampak seolah-olah seluruh Web berada di

dalam basis pengetahuan. Ini adalah contoh teknik umum yang disebut lampiran prosedural, di mana predikat dan fungsi tertentu dapat ditangani dengan metode bertujuan khusus.

Membandingkan Penawaran

Mari kita asumsikan bahwa proses penalaran pada bagian sebelumnya telah menghasilkan serangkaian halaman penawaran untuk kueri "laptop" kita. Untuk membandingkan penawaran tersebut, agen harus mengekstrak informasi yang relevan—harga, kecepatan, ukuran disk, berat, dan sebagainya—dari halaman penawaran. Ini bisa menjadi tugas yang sulit dengan halaman Web yang sebenarnya, karena semua alasan yang disebutkan sebelumnya. Cara umum untuk menangani masalah ini adalah dengan menggunakan program yang disebut wrapper untuk mengekstrak informasi dari suatu halaman. Untuk saat ini, kita asumsikan bahwa wrapper ada, dan ketika diberikan halaman dan basis pengetahuan, wrapper menambahkan pernyataan ke basis pengetahuan. Biasanya, hierarki wrapper akan diterapkan ke suatu halaman: yang sangat umum untuk mengekstrak tanggal dan harga, yang lebih spesifik untuk mengekstrak atribut untuk produk yang berhubungan dengan komputer, dan jika perlu, wrapper khusus situs yang mengetahui format toko tertentu. Diberikan halaman di situs example.com dengan teks

IBM ThinkBook 970. Our price: Rp 3.990.000

diikuti dengan berbagai spesifikasi teknis, kami ingin pembungkus untuk mengekstrak informasi seperti berikut:

$$\begin{aligned} \exists c, offer \ c \in LaptopComputers \wedge offer \in ProductOffers \wedge \\ Manufacturer(c, IBM) \wedge Model(c, ThinkBook970) \wedge \\ ScreenSize(c, Inches(14)) \wedge ScreenType(c, ColorLCD) \wedge \\ MemorySize(c, Gigabytes(2)) \wedge CPUSpeed(c, GHz(1.2)) \wedge \\ OfferedProduct(offer, c) \wedge Store(offer, GenStore) \wedge \\ URL(offer, "example.com/computers/34356.html") \wedge \\ Price(offer, Rp(3.990.000)) \wedge Date(offer, Today). \end{aligned}$$

Contoh ini menggambarkan beberapa masalah yang muncul ketika kita menganggap serius tugas rekayasa pengetahuan untuk transaksi komersial. Misalnya, perhatikan bahwa harga merupakan atribut dari penawaran, bukan produk itu sendiri. Hal ini penting karena penawaran di toko tertentu dapat berubah dari hari ke hari bahkan untuk laptop yang sama; untuk beberapa kategori—seperti rumah dan lukisan—objek yang sama bahkan dapat ditawarkan secara bersamaan oleh perantara yang berbeda dengan harga yang berbeda. Masih ada lebih banyak komplikasi yang belum kita tangani, seperti kemungkinan bahwa harga bergantung pada metode pembayaran dan kualifikasi pembeli untuk mendapatkan diskon tertentu. Tugas terakhir adalah membandingkan penawaran yang telah diekstraksi. Misalnya, pertimbangkan tiga penawaran ini:

A : 1.4 GHz CPU, 2GB RAM, 250 GB disk, Rp 2.990.000 .

B : 1.2 GHz CPU, 4GB RAM, 350 GB disk, Rp5.000.000 .

C : 1.2 GHz CPU, 2GB RAM, 250 GB disk, Rp 3.990.000 .

C didominasi oleh *A*; artinya, *A* lebih murah dan lebih cepat, dan keduanya sama saja. Secara umum, *X* mendominasi *Y* jika *X* memiliki nilai yang lebih baik pada setidaknya satu atribut, dan tidak lebih buruk pada atribut apa pun. Namun, baik *A* maupun *B* tidak mendominasi yang lain. Untuk memutuskan mana yang lebih baik, kita perlu mengetahui bagaimana pembeli menimbang kecepatan CPU dan harga terhadap memori dan ruang disk. Agen belanja kita hanya akan mengembalikan daftar semua penawaran yang tidak didominasi yang memenuhi deskripsi pembeli. Dalam contoh ini, baik *A* maupun *B* tidak didominasi. Perhatikan bahwa hasil ini bergantung pada asumsi bahwa setiap orang lebih menyukai harga yang lebih murah, prosesor yang lebih cepat, dan penyimpanan yang lebih besar. Beberapa atribut, seperti ukuran layar pada notebook, bergantung pada preferensi khusus pengguna (portabilitas versus visibilitas); untuk ini, agen belanja hanya perlu bertanya kepada pengguna.

Agen belanja yang kami jelaskan di sini sederhana; banyak penyempurnaan yang mungkin dilakukan. Namun, ia memiliki cukup kemampuan sehingga dengan pengetahuan khusus domain yang tepat, ia benar-benar dapat berguna bagi pembeli. Karena konstruksinya deklaratifnya, ia dapat dengan mudah diperluas ke aplikasi yang lebih kompleks. Inti utama dari bagian ini adalah untuk menunjukkan bahwa beberapa representasi pengetahuan—khususnya, hierarki produk—diperlukan untuk agen semacam itu, dan bahwa begitu kita memiliki beberapa pengetahuan dalam bentuk ini, sisanya akan mengikuti secara alami.

3.8 RINGKASAN

Dengan menggali lebih dalam tentang bagaimana seseorang merepresentasikan berbagai pengetahuan, kami berharap kami telah memberikan pembaca suatu pemahaman tentang bagaimana basis pengetahuan nyata dibangun dan perasaan terhadap isu-isu filosofis menarik yang muncul. Poin-poin utamanya adalah sebagai berikut:

- Representasi pengetahuan berskala besar memerlukan ontologi tujuan umum untuk mengatur dan menghubungkan berbagai domain pengetahuan tertentu.
- Ontologi tujuan umum perlu mencakup berbagai pengetahuan dan harus mampu, pada prinsipnya, menangani domain apa pun.
- Membangun ontologi tujuan umum yang besar merupakan tantangan signifikan yang belum sepenuhnya terwujud, meskipun kerangka kerja saat ini tampaknya cukup kuat.
- Kami menyajikan ontologi atas berdasarkan kategori dan kalkulus kejadian. Kami membahas kategori, subkategori, bagian, objek terstruktur, pengukuran, zat, kejadian, waktu dan ruang, perubahan, dan kepercayaan.
- Macam-macam alamiah tidak dapat didefinisikan sepenuhnya dalam logika, tetapi sifat-sifat macam-macam alamiah dapat direpresentasikan.

- Tindakan, peristiwa, dan waktu dapat direpresentasikan baik dalam kalkulus situasi maupun dalam representasi yang lebih ekspresif seperti kalkulus peristiwa. Representasi semacam itu memungkinkan agen untuk menyusun rencana melalui inferensi logis.
- Kami menyajikan analisis terperinci tentang domain belanja Internet, dengan menggunakan ontologi umum dan menunjukkan bagaimana pengetahuan domain dapat digunakan oleh agen belanja.
- Sistem representasi tujuan khusus, seperti jaringan semantik dan logika deskripsi, telah dirancang untuk membantu dalam mengatur hierarki kategori. Pewarisan merupakan bentuk inferensi penting, yang memungkinkan properti objek disimpulkan dari keanggotaannya dalam kategori.
- Asumsi dunia tertutup, sebagaimana diterapkan dalam program logika, menyediakan cara sederhana untuk menghindari keharusan menentukan banyak informasi negatif. Asumsi ini paling baik ditafsirkan sebagai default yang dapat ditimpa oleh informasi tambahan.
- Logika nonmonotonik, seperti pembatasan dan logika default, dimaksudkan untuk menangkap penalaran default secara umum.
- Sistem pemeliharaan kebenaran menangani pembaruan dan revisi pengetahuan secara efisien.

BAB 4

MENGUKUR KETIDAKPASTIAN

4.1 BERTINDAK DI BAWAH KETIDAKPASTIAN

Agen mungkin perlu menangani ketidakpastian, baik karena observabilitas parsial, nondeterminisme, atau kombinasi keduanya. Agen mungkin tidak pernah tahu dengan pasti keadaannya saat ini atau di mana ia akan berakhir setelah serangkaian tindakan.

Kita telah melihat agen pemecahan masalah dan agen logis yang dirancang untuk menangani ketidakpastian dengan melacak keadaan keyakinan—representasi dari kumpulan semua kemungkinan keadaan dunia yang mungkin terjadi—dan menghasilkan rencana kontinjensi yang menangani setiap kemungkinan kejadian yang mungkin dilaporkan oleh sensornya selama eksekusi. Meskipun memiliki banyak kelebihan, pendekatan ini memiliki kekurangan yang signifikan jika diambil secara harfiah sebagai resep untuk membuat program agen:

- Saat menafsirkan informasi sensor parsial, agen logis harus mempertimbangkan setiap penjelasan yang mungkin secara logis untuk pengamatan, tidak peduli seberapa tidak mungkin. Hal ini mengarah pada representasi status keyakinan yang besar dan kompleks yang mustahil.
- Rencana kontinjensi yang benar yang menangani setiap kemungkinan dapat tumbuh sangat besar dan harus mempertimbangkan kontinjensi yang tidak mungkin terjadi.
- Terkadang tidak ada rencana yang dijamin untuk mencapai tujuan—namun agen harus bertindak. Agen harus memiliki cara untuk membandingkan manfaat rencana yang tidak dijamin.

Misalnya, anggaplah taksi otomatis memiliki tujuan untuk mengantarkan penumpang ke bandara tepat waktu. Agen membentuk rencana, A_{90} , yang melibatkan meninggalkan rumah 90 menit sebelum keberangkatan pesawat dan mengemudi dengan kecepatan yang wajar. Meskipun bandara hanya berjarak sekitar 5 mil, agen taksi yang logis tidak akan dapat menyimpulkan dengan pasti bahwa "Rencana A_{90} akan membawa kita ke bandara tepat waktu." Sebaliknya, ia mencapai kesimpulan yang lebih lemah "Rencana A_{90} akan membawa kita ke bandara tepat waktu, selama mobil tidak mogok atau kehabisan bensin, dan saya tidak mengalami kecelakaan, dan tidak ada kecelakaan di anjungan, dan pesawat tidak berangkat lebih awal, dan tidak ada meteorit yang menabrak mobil, dan" Tidak satu pun dari kondisi ini dapat disimpulkan dengan pasti, jadi keberhasilan rencana tersebut tidak dapat disimpulkan. Ini adalah masalah kualifikasi, yang sejauh ini belum kita lihat solusinya.

Meskipun demikian, dalam beberapa hal A_{90} sebenarnya adalah hal yang benar untuk dilakukan. Apa yang kita maksud dengan ini? Yang kita maksud adalah bahwa dari semua rencana yang dapat dilaksanakan, A_{90} diharapkan dapat memaksimalkan ukuran kinerja agen (di mana ekspektasi tersebut relatif terhadap pengetahuan agen tentang lingkungan). Ukuran kinerja tersebut mencakup tiba di bandara tepat waktu untuk penerbangan, menghindari

waktu tunggu yang lama dan tidak produktif di bandara, dan menghindari tilang di sepanjang jalan. Pengetahuan agen tidak dapat menjamin hasil apa pun untuk A_{90} , tetapi dapat memberikan tingkat keyakinan tertentu bahwa hasil tersebut akan tercapai. Rencana lain, seperti A_{180} , dapat meningkatkan keyakinan agen bahwa ia akan tiba di bandara tepat waktu, tetapi juga meningkatkan kemungkinan waktu tunggu yang lama. Oleh karena itu, hal yang benar untuk dilakukan—keputusan rasional—bergantung pada kepentingan relatif berbagai tujuan dan kemungkinan, serta tingkat pencapaian tujuan tersebut. Sisa bagian ini mengasah gagasan-gagasan ini, sebagai persiapan untuk pengembangan teori-teori umum tentang penalaran yang tidak pasti dan keputusan-keputusan rasional yang kami sajikan dalam bab ini dan bab-bab berikutnya.

Merangkum Ketidakpastian

Mari kita pertimbangkan sebuah contoh penalaran yang tidak pasti: mendiagnosis toothache seorang pasien tooth. Diagnosis—entah untuk pengobatan, perbaikan mobil, atau apa pun—hampir selalu melibatkan ketidakpastian. Mari kita coba menulis aturan-aturan untuk diagnosis tooth menggunakan logika proposisional, sehingga kita dapat melihat bagaimana pendekatan logis tersebut gagal. Pertimbangkan aturan sederhana berikut ini:

$$\text{Toothache} \Rightarrow \text{Cavity}.$$

Masalahnya adalah aturan ini salah. Tidak semua pasien toothache memiliki cavity; beberapa dari mereka memiliki penyakit gusi, abses, atau salah satu dari beberapa masalah lainnya:

$$\text{Toothache} \Rightarrow \text{Cavity} \vee \text{GumProblem} \vee \text{Abscess} \dots$$

Sayangnya, untuk menjadikan aturan tersebut benar, kita harus menambahkan daftar kemungkinan masalah yang hampir tak terbatas. Kita dapat mencoba mengubah aturan tersebut menjadi aturan kausal:

$$\text{Cavity} \Rightarrow \text{Toothache}.$$

Namun, aturan ini juga tidak benar; tidak semua cavity menyebabkan rasa sakit. Satu-satunya cara untuk memperbaiki aturan tersebut adalah dengan membuatnya lengkap secara logis: dengan melengkapi sisi kiri dengan semua kualifikasi yang dibutuhkan agar cavity dapat menyebabkan toothache. Dengan demikian, mencoba menggunakan logika untuk mengatasi domain seperti diagnosis medis gagal karena tiga alasan utama:

- **Kemalasan:** Terlalu banyak pekerjaan untuk mencantumkan rangkaian lengkap anteseden atau konsekuensi yang dibutuhkan untuk memastikan aturan tanpa pengecualian dan terlalu sulit untuk menggunakan aturan tersebut.
- **Ketidaktahuan teoritis:** Ilmu kedokteran tidak memiliki teori lengkap untuk domain tersebut.

- **Ketidaktahuan praktis:** Bahkan jika kita mengetahui semua aturan, kita mungkin tidak yakin tentang pasien tertentu karena tidak semua tes yang diperlukan telah atau dapat dijalankan.

Hubungan antara toothache dan cavity hanyalah bukan konsekuensi logis dalam kedua arah. Ini adalah tipikal domain medis, serta sebagian besar domain penilaian lainnya: hukum, bisnis, desain, perbaikan mobil, berkebun, kencan, dan sebagainya. Pengetahuan agen paling-paling hanya dapat memberikan tingkat kepercayaan pada kalimat-kalau yang relevan. Alat utama kita untuk berurusan dengan tingkat kepercayaan adalah teori probabilitas. Komitmen ontologis logika dan teori probabilitas adalah sama—bahwa dunia terdiri dari fakta-fakta yang ada atau tidak ada dalam kasus tertentu—tetapi komitmen epistemologisnya berbeda: agen logis percaya setiap kalimat itu benar atau salah atau tidak memiliki pendapat, sedangkan agen probabilistik mungkin memiliki tingkat kepercayaan numerik antara 0 (untuk kalimat yang pasti salah) dan 1 (pasti benar).

Probabilitas menyediakan cara untuk meringkas ketidakpastian yang berasal dari kemalasan dan ketidaktahuan kita, dengan demikian memecahkan masalah kualifikasi. Kita mungkin tidak tahu pasti apa yang menimpa pasien tertentu, tetapi kita percaya bahwa ada, katakanlah, peluang 80%—yaitu, probabilitas 0,8—bahwa pasien yang toothache memiliki cavity. Artinya, kami memperkirakan bahwa dari semua situasi yang tidak dapat dibedakan dari situasi saat ini sejauh pengetahuan kami, pasien akan memiliki cavity pada 80% dari mereka. Keyakinan ini dapat diperoleh dari data statistik—80% dari pasien toothache yang ditangani sejauh ini memiliki cavity—atau dari beberapa pengetahuan umum tentang kedokteran tooth, atau dari kombinasi sumber bukti.

Satu hal yang membingungkan adalah bahwa pada saat diagnosis kami, tidak ada ketidakpastian di dunia nyata: pasien memiliki cavity atau tidak. Jadi, apa artinya mengatakan probabilitas cavity adalah 0,8? Bukankah seharusnya 0 atau 1? Jawabannya adalah bahwa pernyataan probabilitas dibuat sehubungan dengan status pengetahuan, bukan sehubungan dengan dunia nyata. Kami mengatakan "Probabilitas pasien memiliki cavity, mengingat dia toothache, adalah 0,8." Jika kemudian kita mengetahui bahwa pasien memiliki riwayat penyakit gusi, kita dapat membuat pernyataan yang berbeda: "Kemungkinan pasien memiliki cavity, mengingat ia memiliki toothache dan riwayat penyakit gusi, adalah 0,4." Jika kita mengumpulkan bukti konklusif lebih lanjut terhadap cavity, kita dapat mengatakan "Kemungkinan pasien memiliki cavity, mengingat semua yang kita ketahui sekarang, hampir 0." Perhatikan bahwa pernyataan-pernyataan ini tidak saling bertentangan; masing-masing adalah pernyataan terpisah tentang status pengetahuan yang berbeda.

Ketidakpastian dan Keputusan Rasional

Pertimbangkan lagi rencana A_{90} untuk pergi ke bandara. Misalkan rencana tersebut memberi kita peluang 97% untuk mengejar penerbangan kita. Apakah ini berarti bahwa itu adalah pilihan yang rasional? Tidak harus: mungkin ada rencana lain, seperti A_{180} , dengan probabilitas yang lebih tinggi. Jika sangat penting untuk tidak ketinggalan penerbangan, maka ada baiknya mengambil risiko menunggu lebih lama di bandara. Bagaimana dengan A_{1440} , rencana yang mengharuskan kita berangkat dari rumah 24 jam sebelumnya? Dalam

kebanyakan situasi, ini bukanlah pilihan yang baik, karena meskipun hampir menjamin kita tiba tepat waktu, rencana ini mengharuskan kita menunggu terlalu lama—belum lagi kemungkinan makanan bandara yang tidak enak.

Untuk membuat pilihan seperti itu, agen pertama-tama harus memiliki preferensi di antara berbagai kemungkinan hasil dari berbagai rencana. Hasil adalah keadaan yang sepenuhnya ditentukan, termasuk faktor-faktor seperti apakah agen tiba tepat waktu dan lamanya waktu menunggu di bandara. Kami menggunakan teori utilitas untuk merepresentasikan dan bernalar dengan preferensi. (Istilah utilitas digunakan di sini dalam pengertian “kualitas yang bermanfaat,” bukan dalam pengertian perusahaan listrik atau perusahaan air.) Teori utilitas menyatakan bahwa setiap keadaan memiliki tingkat kegunaan, atau utilitas, bagi seorang agen dan bahwa agen akan lebih memilih keadaan dengan utilitas yang lebih tinggi.

Kegunaan suatu keadaan relatif terhadap agen. Misalnya, kegunaan suatu keadaan di mana Putih telah melakukan skakmat terhadap Hitam dalam permainan catur jelas tinggi bagi agen yang memainkan Putih, tetapi rendah bagi agen yang memainkan Hitam. Namun, kita tidak dapat sepenuhnya berpegang pada skor 1, 1/2, dan 0 yang ditentukan oleh aturan catur turnamen—beberapa pemain (termasuk penulis) mungkin senang dengan hasil seri melawan juara dunia, sedangkan pemain lain (termasuk mantan juara dunia) mungkin tidak. Tidak ada yang dapat menjelaskan selera atau preferensi: Anda mungkin berpikir bahwa agen yang lebih menyukai es krim permen karet jalapeno daripada cokelat keping adalah aneh atau bahkan salah arah, tetapi Anda tidak dapat mengatakan bahwa agen tersebut tidak rasional. Fungsi kegunaan dapat menjelaskan serangkaian preferensi apa pun—aneh atau biasa, mulia atau menyimpang. Perhatikan bahwa kegunaan dapat menjelaskan altruisme, hanya dengan memasukkan kesejahteraan orang lain sebagai salah satu faktornya.

Preferensi, sebagaimana diungkapkan oleh kegunaan, digabungkan dengan probabilitas dalam teori umum keputusan rasional yang disebut teori keputusan:

$$\textit{Decision theory} = \textit{probability theory} + \textit{utility theory}.$$

Ide mendasar dari teori keputusan adalah bahwa seorang agen adalah rasional jika dan hanya jika ia memilih tindakan yang menghasilkan utilitas yang diharapkan tertinggi, yang dirata-ratakan dari semua kemungkinan hasil tindakan tersebut. Ini disebut prinsip utilitas yang diharapkan maksimum (MEU). Perhatikan bahwa "diharapkan" mungkin tampak seperti istilah hipotetis yang samar, tetapi seperti yang digunakan di sini, ia memiliki makna yang tepat: ia berarti "rata-rata," atau "rata-rata statistik" dari hasil, yang dibobot dengan probabilitas hasil.

Gambar 4.1 menggambarkan struktur agen yang menggunakan teori keputusan untuk memilih tindakan. Agen tersebut identik, pada tingkat abstrak, yang mempertahankan status keyakinan yang mencerminkan sejarah persepsi hingga saat ini. Perbedaan utamanya adalah bahwa status keyakinan agen yang berteori keputusan tidak hanya mewakili kemungkinan untuk status dunia tetapi juga probabilitasnya. Mengingat kondisi keyakinan, agen dapat membuat prediksi probabilistik atas hasil tindakan dan karenanya memilih tindakan dengan

utilitas yang diharapkan tertinggi. Bab ini dan bab berikutnya berkonsentrasi pada tugas merepresentasikan dan menghitung dengan informasi probabilistik secara umum. Bab 15 membahas metode untuk tugas-tugas khusus merepresentasikan dan memperbarui kondisi keyakinan dari waktu ke waktu dan memprediksi lingkungan. Bab 16 membahas teori utilitas secara lebih mendalam, dan Bab 17 mengembangkan algoritme untuk merencanakan urutan tindakan dalam lingkungan yang tidak pasti.

4.2 NOTASI PROBABILITAS DASAR

Agar agen kita merepresentasikan dan menggunakan informasi probabilistik, kita memerlukan bahasa formal. Bahasa teori probabilitas secara tradisional bersifat informal, ditulis oleh matematikawan manusia kepada matematikawan manusia lainnya. Lampiran A mencakup pengantar standar untuk teori probabilitas dasar; di sini, kita mengambil pendekatan yang lebih sesuai dengan kebutuhan AI dan lebih konsisten dengan konsep logika formal.

function DT – AGENT(*percept*) **returns** an *action*
persistent: *belief_state*, probabilistic beliefs about the current state of the world
action, the agent's action

update *belief_state* based on *action* and *percept*
 calculate outcome probabilities for actions,
 given action descriptions and current *belief_state*
 select *action* with highest expected utility
 given probabilities of outcomes and utility information
return *action*

Gambar 4.1 Agen teoretis keputusan yang memilih tindakan rasional.

Tentang Apa Probabilitas Itu

Seperti pernyataan logis, pernyataan probabilistik adalah tentang kemungkinan dunia. Sementara pernyataan logis mengatakan kemungkinan dunia mana yang benar-benar dikesampingkan (semua yang pernyataannya salah), pernyataan probabilistik berbicara tentang seberapa besar kemungkinan berbagai dunia itu. Dalam teori probabilitas, himpunan semua kemungkinan dunia disebut ruang sampel. Kemungkinan dunia saling eksklusif dan menyeluruh dua kemungkinan dunia tidak mungkin sama-sama terjadi, dan satu kemungkinan dunia pasti terjadi. Misalnya, jika kita akan melempar dua dadu (yang dapat dibedakan), ada 36 kemungkinan dunia yang perlu dipertimbangkan: (1,1), (1,2), .. ., (6,6). Huruf Yunani Ω (omega huruf kapital) digunakan untuk merujuk ke ruang sampel, dan ω (omega huruf kecil) merujuk ke elemen ruang, yaitu kemungkinan dunia tertentu.

Model probabilitas yang ditentukan secara penuh mengaitkan probabilitas numerik $P(\omega)$ dengan setiap kemungkinan dunia. Aksioma dasar teori probabilitas menyatakan bahwa setiap kemungkinan dunia memiliki probabilitas antara 0 dan 1 dan bahwa total probabilitas dari himpunan kemungkinan dunia adalah 1:

$$0 \leq P(\omega) \leq 1 \text{ for every } \omega \text{ and } \sum_{\omega \in \Omega} P(\omega) = 1. \quad (13.1)$$

Misalnya, jika kita berasumsi bahwa setiap dadu adil dan lemparan dadu tidak saling mengganggu, maka setiap kemungkinan dunia $(1,1)$, $(1,2)$, ..., $(6,6)$ memiliki probabilitas $1/36$. Di sisi lain, jika dadu bersekolongkol untuk menghasilkan angka yang sama, maka dunia $(1,1)$, $(2,2)$, $(3,3)$, dst., mungkin memiliki probabilitas yang lebih tinggi, sehingga dunia lainnya memiliki probabilitas yang lebih rendah.

Pernyataan dan pertanyaan probabilistik biasanya tidak membahas kemungkinan dunia tertentu, tetapi membahas kumpulan kemungkinan tersebut. Misalnya, kita mungkin tertarik pada kasus di mana dua dadu berjumlah 11, kasus di mana dadu yang dilempar berjumlah dua, dan seterusnya. Dalam teori probabilitas, kumpulan ini disebut peristiwa. Dalam AI, kumpulan tersebut selalu dijelaskan oleh proposisi dalam bahasa formal. Untuk setiap proposisi, himpunan yang sesuai hanya berisi kemungkinan dunia tempat proposisi tersebut berlaku. Probabilitas yang terkait dengan proposisi didefinisikan sebagai jumlah probabilitas dunia tempat proposisi tersebut berlaku:

$$\text{For any proposition } \phi, P(\phi) = \sum_{\omega \in \phi} P(\omega). \quad (13.2)$$

Misalnya, ketika melempar dadu yang adil, kita memiliki $P(\text{Total} = 11) = P((5,6)) + P((6,5)) = 1/36 + 1/36 = 1/18$. Perhatikan bahwa teori probabilitas tidak memerlukan pengetahuan lengkap tentang probabilitas setiap kemungkinan dunia. Misalnya, jika kita percaya dadu bersekolongkol untuk menghasilkan angka yang sama, kita mungkin menyatakan bahwa $P(\text{doubles}) = 1/4$ tanpa mengetahui apakah dadu lebih menyukai ganda 6 daripada ganda 2. Sama seperti pernyataan logis, pernyataan ini membatasi model probabilitas yang mendasarinya tanpa sepenuhnya menentukannya.

Probabilitas seperti $P(\text{Total} = 11)$ dan $P(\text{doubles})$ disebut probabilitas tanpa syarat atau sebelumnya (dan terkadang hanya "prior" untuk kependekannya); mereka merujuk pada derajat kepercayaan pada proposisi tanpa adanya informasi lain. Namun, sebagian besar waktu, kita memiliki beberapa informasi, biasanya disebut bukti, yang telah terungkap. Misalnya, dadu pertama mungkin sudah menunjukkan angka 5 dan kita menunggu dengan napas tertahan hingga dadu lainnya berhenti berputar. Dalam kasus tersebut, kita tidak tertarik pada probabilitas tanpa syarat untuk melempar dadu ganda, tetapi probabilitas bersyarat atau posterior (atau singkatnya "posterior") untuk melempar dadu ganda mengingat dadu pertama adalah angka 5. Probabilitas ini ditulis $P(\text{doubles} | \text{Die}_1 = 5)$, di mana " | " diucapkan "given." Demikian pula, jika saya pergi ke dokter tooth untuk pemeriksaan rutin, probabilitas $P(\text{cavity}) = 0,2$ mungkin menarik; tetapi jika saya pergi ke dokter tooth karena saya toothache, maka $P(\text{cavity} | \text{toothache}) = 0,6$ yang penting. Perhatikan bahwa prioritas " | " sedemikian rupa sehingga setiap ekspresi dalam bentuk $P(\dots | \dots)$ selalu berarti $P(\dots | (\dots))$.

Penting untuk memahami bahwa $P(cavity) = 0,2$ masih berlaku setelah *toothache* diamati; hanya saja tidak terlalu berguna. Saat membuat keputusan, agen perlu mengondisikan semua bukti yang telah diamatinya. Penting juga untuk memahami perbedaan antara pengkondisian dan implikasi logis. Pernyataan bahwa $P(cavity|toothache) = 0,6$ tidak berarti "Setiap kali *toothache* benar, simpulkan bahwa *cavity* benar dengan probabilitas 0,6" melainkan berarti "Setiap kali *toothache* benar dan kita tidak memiliki informasi lebih lanjut, simpulkan bahwa *cavity* benar dengan probabilitas 0,6." Kondisi tambahan itu penting; misalnya, jika kita memiliki informasi lebih lanjut bahwa dokter *tooth* tidak menemukan *cavity*, kita pasti tidak ingin menyimpulkan bahwa *cavity* itu benar dengan probabilitas 0,6; sebaliknya kita perlu menggunakan $P(cavity|toothache \wedge \neg cavity) = 0$.

Secara matematis, probabilitas bersyarat didefinisikan dalam istilah probabilitas tak bersyarat sebagai berikut: untuk setiap proposisi a dan b , kita memiliki

$$P(a | b) = \frac{P(a \wedge b)}{P(b)} \quad (13.3)$$

yang berlaku setiap kali $P(b) > 0$. Misalnya,

$$P(doubles | Die_1 = 5) = \frac{P(doubles \wedge Die_1 = 5)}{P(Die_1 = 5)}$$

Definisi tersebut masuk akal jika Anda ingat bahwa mengamati b mengesampingkan semua kemungkinan dunia di mana b salah, sehingga menyisakan satu set yang total probabilitasnya hanya $P(b)$. Dalam set tersebut, dunia a memenuhi $a \wedge b$ dan membentuk pecahan $P(a \wedge b)/P(b)$.

Definisi probabilitas bersyarat, Persamaan (4.3), dapat ditulis dalam bentuk berbeda yang disebut aturan produk:

$$P(a \wedge b) = P(a | b)P(b),$$

Aturan perkalian mungkin lebih mudah diingat: aturan ini berasal dari fakta bahwa, agar a dan b benar, kita perlu b benar, dan kita juga perlu a benar jika b diberikan.

Bahasa Proposisi dalam Pernyataan Probabilitas

Dalam bab ini dan bab berikutnya, proposisi yang menggambarkan himpunan kemungkinan dunia ditulis dalam notasi yang menggabungkan elemen logika proposisional dan notasi pemenuhan kendala. Dalam terminologi, ini adalah representasi terfaktor, di mana kemungkinan dunia diwakili oleh himpunan pasangan variabel/nilai.

Variabel dalam teori probabilitas disebut variabel acak dan namanya dimulai dengan huruf kapital. Jadi, dalam contoh dadu, *Total* dan *Die₁* adalah variabel acak. Setiap variabel acak memiliki domain—himpunan nilai yang mungkin yang dapat diambilnya. Domain *Total*

untuk dua dadu adalah himpunan $\{2, \dots, 12\}$ dan domain Die_1 adalah $\{1, \dots, 6\}$. Variabel acak Boolean memiliki domain $\{true, false\}$ (perhatikan bahwa nilai selalu huruf kecil); misalnya, proposisi bahwa angka ganda digulirkan dapat ditulis sebagai $Doubles = true$. Berdasarkan konvensi, proposisi bentuk $A = true$ disingkat menjadi a , sedangkan $A = false$ disingkat menjadi $\neg a$. (Penggunaan *doubles*, *cavity*, dan *toothache* pada bagian sebelumnya adalah singkatan dari jenis ini.) Seperti dalam CSP, domain dapat berupa kumpulan token yang berubah-ubah; kita dapat memilih domain *Age* menjadi $\{juvenile, teen, adult\}$ dan domain *Weather* mungkin menjadi $\{sunny, rain, cloudy, snow\}$. Ketika tidak ada ambiguitas yang mungkin terjadi, adalah umum untuk menggunakan nilai dengan sendirinya untuk mewakili proposisi bahwa variabel tertentu memiliki nilai itu; dengan demikian, *sunny* dapat mewakili $Weather = sunny$.

Semua contoh sebelumnya memiliki domain terbatas. Variabel juga dapat memiliki domain tak terbatas—baik diskrit (seperti bilangan bulat) maupun kontinu (seperti bilangan riil). Untuk variabel apa pun dengan domain terurut, pertidaksamaan juga diperbolehkan, seperti $NumberOfAtomsInUniverse \geq 10^{70}$.

Terakhir, kita dapat menggabungkan berbagai macam proposisi elementer ini (termasuk bentuk singkatan untuk variabel Boolean) dengan menggunakan konektif logika proposisional. Misalnya, kita dapat menyatakan “Probabilitas pasien memiliki *cavity*, mengingat dia adalah remaja yang tidak *toothache*, adalah 0,1” sebagai berikut:

$$P(cavity \mid \neg toothache \wedge teen) = 0.1.$$

Terkadang kita ingin berbicara tentang probabilitas semua nilai yang mungkin dari sebuah variabel acak. Kita dapat menulis:

$$\begin{aligned} P(Weather = sunny) &= 0.6 \\ P(Weather = rain) &= 0.1 \\ P(Weather = cloudy) &= 0.29 \\ P(Weather = snow) &= 0.01, \end{aligned}$$

tapi sebagai singkatan kami akan mengizinkan

$$P(Weather) = \langle 0.6, 0.1, 0.29, 0.01 \rangle,$$

di mana P yang dicetak tebal menunjukkan bahwa hasilnya adalah vektor angka, dan di mana kita mengasumsikan urutan yang telah ditentukan sebelumnya $\langle cerah, hujan, cloudy, salju \rangle$ pada domain *Weather*. Kita katakan bahwa pernyataan P mendefinisikan distribusi probabilitas untuk variabel acak *Weather*. Notasi P juga digunakan untuk distribusi kondisional: $P(X \mid Y)$ memberikan nilai $P(X = x_i \mid Y = y_j)$ untuk setiap kemungkinan pasangan i, j .

Untuk variabel kontinu, tidak mungkin untuk menulis seluruh distribusi sebagai vektor, karena ada banyak nilai yang tak terhingga. Sebaliknya, kita dapat mendefinisikan probabilitas bahwa variabel acak mengambil beberapa nilai x sebagai fungsi parameter dari x . Misalnya, kalimat

$$P(\text{NoonTemp} = x) = \text{Uniform}_{[18C, 26C]}(x)$$

menyatakan keyakinan bahwa suhu pada siang hari terdistribusi secara merata antara 18 dan 26 derajat Celsius. Kita menyebutnya fungsi kerapatan probabilitas. Fungsi kerapatan probabilitas (kadang-kadang disebut pdf) berbeda maknanya dari distribusi diskrit. Mengatakan bahwa kerapatan probabilitas seragam dari 18C hingga 26C berarti ada peluang 100% bahwa suhu akan turun di suatu tempat di wilayah selebar 8C dan peluang 50% bahwa suhu akan turun di wilayah selebar 4C, dan seterusnya. Kita menulis kerapatan probabilitas untuk variabel acak kontinu X pada nilai x sebagai $P(X = x)$ atau hanya $P(x)$; definisi intuitif $P(x)$ adalah probabilitas bahwa X berada dalam wilayah yang sangat kecil yang dimulai pada x , dibagi dengan lebar wilayah tersebut:

$$P(x) = \lim_{dx \rightarrow 0} P(x \leq X \leq x + dx) / dx.$$

Untuk NoonTemp kami memiliki

$$P(\text{NoonTemp} = x) = \text{Uniform}_{[18C, 26C]}(x) = \begin{cases} \frac{1}{8C} & \text{if } 18C \leq x \leq 26C \\ 0 & \text{otherwise} \end{cases}$$

di mana C adalah singkatan dari celcius (bukan untuk konstanta). Dalam $P(\text{NoonTemp} = 20.18C) = 1 \frac{1}{8C}$, perhatikan bahwa $\frac{1}{8C}$ bukanlah probabilitas, itu adalah kerapatan probabilitas. Probabilitas bahwa *NoonTemp* tepat 20.18C adalah nol, karena 20.18C adalah wilayah dengan lebar 0. Beberapa penulis menggunakan simbol yang berbeda untuk distribusi diskrit dan fungsi kerapatan; kami menggunakan P dalam kedua kasus, karena kebingungan jarang muncul dan persamaan biasanya identik. Perhatikan bahwa probabilitas adalah angka tanpa satuan, sedangkan fungsi kerapatan diukur dengan satuan, dalam hal ini derajat resiprokal.

Selain distribusi pada variabel tunggal, kita perlu notasi untuk distribusi pada beberapa variabel. Koma digunakan untuk ini. Misalnya, $P(\text{Weather}, \text{Cavity})$ menunjukkan probabilitas semua kombinasi nilai *Weather* dan *Cavity*. Ini adalah tabel probabilitas 4×2 yang disebut distribusi probabilitas gabungan *Weather* dan *Cavity*. Kita juga dapat mencampur variabel dengan dan tanpa nilai; $P(\text{sunny}, \text{Cavity})$ akan menjadi vektor dua elemen yang memberikan probabilitas hari cerah dengan *Cavity* dan hari cerah tanpa *Cavity*. Notasi P membuat ekspresi tertentu jauh lebih ringkas daripada yang seharusnya. Misalnya,

aturan perkalian untuk semua nilai *Weather* dan *Cavity* yang mungkin dapat ditulis sebagai persamaan tunggal:

$$P(\text{Weather}, \text{Cavity}) = P(\text{Weather} | \text{Cavity})P(\text{Cavity}),$$

alih-alih seperti persamaan $4 \times 2 = 8$ berikut (menggunakan singkatan *W* dan *C*):

$$\begin{aligned} P(W = \text{sunny} \wedge C = \text{true}) &= P(W = \text{sunny} | C = \text{true}) P(C = \text{true}) \\ P(W = \text{rain} \wedge C = \text{true}) &= P(W = \text{rain} | C = \text{true}) P(C = \text{true}) \\ P(W = \text{cloudy} \wedge C = \text{true}) &= P(W = \text{cloudy} | C = \text{true}) P(C = \text{true}) \\ P(W = \text{snow} \wedge C = \text{true}) &= P(W = \text{snow} | C = \text{true}) P(C = \text{true}) \\ P(W = \text{sunny} \wedge C = \text{false}) &= P(W = \text{sunny} | C = \text{false}) P(C = \text{false}) \\ P(W = \text{rain} \wedge C = \text{false}) &= P(W = \text{rain} | C = \text{false}) P(C = \text{false}) \\ P(W = \text{cloudy} \wedge C = \text{false}) &= P(W = \text{cloudy} | C = \text{false}) P(C = \text{false}) \\ P(W = \text{snow} \wedge C = \text{false}) &= P(W = \text{snow} | C = \text{false}) P(C = \text{false}). \end{aligned}$$

Sebagai kasus degeneratif, $P(\text{sunny}, \text{Cavity})$ tidak memiliki variabel dan dengan demikian merupakan vektor satu elemen yang merupakan probabilitas hari cerah dengan cavity, yang juga dapat ditulis sebagai $P(\text{sunny}, \text{Cavity})$ atau $P(\text{sunny} \wedge \text{Cavity})$. Kita terkadang akan menggunakan notasi P untuk memperoleh hasil tentang nilai P individual, dan ketika kita mengatakan " $P(\text{sunny}) = 0,6$ ", itu sebenarnya adalah singkatan dari " $P(\text{sunny})$ adalah vektor satu elemen $\langle 0,6 \rangle$, yang berarti bahwa $P(\text{sunny}) = 0,6$."

Sekarang kita telah mendefinisikan sintaksis untuk proposisi dan pernyataan probabilitas dan kita telah memberikan bagian semantik: Persamaan (4.2) mendefinisikan probabilitas proposisi sebagai jumlah probabilitas dunia tempat proposisi itu berlaku. Untuk melengkapi semantik, kita perlu mengatakan apa dunia-dunia itu dan bagaimana menentukan apakah proposisi berlaku di dunia. Kami meminjam bagian ini langsung dari semantik logika proposisional, sebagai berikut. Dunia yang mungkin didefinisikan sebagai penugasan nilai untuk semua variabel acak yang dipertimbangkan. Mudah untuk melihat bahwa definisi ini memenuhi persyaratan dasar bahwa dunia yang mungkin harus saling eksklusif dan lengkap. Misalnya, jika variabel acaknya adalah *Cavity*, *Toothache*, dan *Weather*, maka ada $2 \times 2 \times 4 = 16$ dunia yang mungkin. Lebih jauh, kebenaran dari setiap proposisi yang diberikan, tidak peduli seberapa rumitnya, dapat ditentukan dengan mudah di dunia tersebut menggunakan definisi kebenaran rekursif yang sama seperti untuk rumus dalam logika proposisional.

Dari definisi dunia yang mungkin sebelumnya, dapat disimpulkan bahwa model probabilitas sepenuhnya ditentukan oleh distribusi gabungan untuk semua variabel acak—yang disebut distribusi probabilitas gabungan penuh. Misalnya, jika variabelnya adalah *Cavity*, *Toothache*, dan *Weather*, maka distribusi gabungan penuh diberikan oleh $P(\text{Cavity}, \text{Toothache}, \text{Weather})$. Distribusi gabungan ini dapat direpresentasikan sebagai tabel $2 \times 2 \times 4$ dengan 16 entri. Karena probabilitas setiap proposisi adalah jumlah dari

kemungkinan dunia, distribusi gabungan penuh pada prinsipnya cukup untuk menghitung probabilitas proposisi apa pun.

Aksioma Probabilitas dan Kewajarannya

Aksioma dasar probabilitas (Persamaan (4.1) dan (13.2)) menyiratkan hubungan tertentu di antara derajat keyakinan yang dapat diberikan pada proposisi yang terkait secara logis. Misalnya, kita dapat memperoleh hubungan yang sudah dikenal antara probabilitas suatu proposisi dan probabilitas negasinya:

$$\begin{aligned}
 P(\neg a) &= \sum_{\omega \in \neg a} P(\omega) \\
 &= \sum_{\omega \in \neg a} P(\omega) + \sum_{\omega \in a} P(\omega) - \sum_{\omega \in a} P(\omega) && \text{dengan Persamaan (4.2)} \\
 &= \sum_{\omega \in \Omega} P(\omega) - \sum_{\omega \in a} P(\omega) && \text{mengelompokkan dua suku pertama dengan (13.1) dan (13.2).} \\
 &= 1 - P(a)
 \end{aligned}$$

Kita juga dapat memperoleh rumus yang terkenal untuk probabilitas disjungsi, yang terkadang disebut prinsip inklusi–eksklusi:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b). \quad (13.4)$$

Aturan ini mudah diingat dengan memperhatikan bahwa kasus-kasus di mana a berlaku, bersama dengan kasus-kasus di mana b berlaku, tentu mencakup semua kasus di mana $a \vee b$ berlaku; tetapi menjumlahkan kedua set kasus menghitung irisannya dua kali, jadi kita perlu mengurangi $P(a \wedge b)$.

Persamaan (4.1) dan (13.4) sering disebut aksioma Kolmogorov untuk menghormati matematikawan Rusia Andrei Kolmogorov, yang menunjukkan cara membangun teori probabilitas lainnya dari fondasi sederhana ini dan cara menangani kesulitan yang disebabkan oleh variabel kontinu. Sementara Persamaan (4.2) memiliki cita rasa definisional, Persamaan (4.4) mengungkapkan bahwa aksioma benar-benar membatasi derajat keyakinan yang dapat dimiliki agen mengenai proposisi yang terkait secara logis. Hal ini serupa dengan fakta bahwa agen logis tidak dapat secara bersamaan mempercayai A , B , dan $\neg(A \wedge B)$, karena tidak ada dunia yang memungkinkan di mana ketiganya benar. Namun, dengan probabilitas, pernyataan tidak merujuk ke dunia secara langsung, tetapi ke status pengetahuan agen itu sendiri. Lalu, mengapa seorang agen tidak dapat memegang serangkaian keyakinan berikut (meskipun keyakinan tersebut melanggar aksioma Kolmogorov)?

$$\begin{aligned}
 P(a) &= 0.4 & P(a \wedge b) &= 0.0 \\
 P(b) &= 0.3 & P(a \vee b) &= 0.8.
 \end{aligned} \quad (13.5)$$

Pertanyaan semacam ini telah menjadi pokok perdebatan sengit selama beberapa dekade antara mereka yang menganjurkan penggunaan probabilitas sebagai satu-satunya bentuk yang sah untuk tingkat keyakinan dan mereka yang menganjurkan pendekatan alternatif.

Salah satu argumen untuk aksioma probabilitas, yang pertama kali dinyatakan pada tahun 1931 oleh Bruno de Finetti, adalah sebagai berikut: Jika seorang agen memiliki tingkat keyakinan tertentu pada proposisi a , maka agen tersebut harus dapat menyatakan peluang di mana ia tidak peduli dengan taruhan untuk atau melawan a . Anggap saja ini sebagai permainan antara dua agen: Agen 1 menyatakan, "tingkat keyakinan saya pada peristiwa a adalah 0,4." Agen 2 kemudian bebas memilih apakah akan bertaruh untuk atau melawan a pada taruhan yang konsisten dengan tingkat keyakinan yang dinyatakan. Artinya, Agen 2 dapat memilih untuk menerima taruhan Agen 1 bahwa a akan terjadi, dengan menawarkan Rp 60.000 melawan Rp 40.000 milik Agen 1. Atau Agen 2 dapat menerima taruhan Agen 1 bahwa $\neg a$ akan terjadi, dengan menawarkan Rp 40.000 melawan Rp 60.000 milik Agen 1. Kemudian kita mengamati hasil dari a , dan siapa pun yang benar akan mendapatkan uangnya. Jika tingkat keyakinan seorang agen tidak secara akurat mencerminkan dunia, maka Anda akan menduga bahwa agen tersebut akan cenderung kehilangan uang dalam jangka panjang kepada agen lawan yang keyakinannya lebih akurat mencerminkan keadaan dunia. Namun de Finetti membuktikan sesuatu yang jauh lebih kuat: Jika Agen 1 menyatakan serangkaian tingkat keyakinan yang melanggar aksioma teori probabilitas maka ada kombinasi taruhan oleh Agen 2 yang menjamin bahwa Agen 1 akan kehilangan uang setiap saat. Misalnya, anggaplah Agen 1 memiliki serangkaian tingkat keyakinan dari Persamaan (4.5). Gambar 4.2 menunjukkan bahwa jika Agen 2 memilih untuk bertaruh Rp 40.000 pada a , Rp 30.000 pada b , dan Rp 20.000 pada $\neg(a \vee b)$, maka Agen 1 selalu kehilangan uang, terlepas dari hasil untuk a dan b . Teorema De Finetti menyiratkan bahwa tidak ada agen rasional yang dapat memiliki keyakinan yang melanggar aksioma probabilitas.

Agen 1		Agen 2		Hasil dan keuntungan untuk Agen 1			
Proposisi	Keyakinan	Taruhan	Taruhan	a, b	$a, \neg b$	$\neg a, b$	$\neg a, \neg b$
a	0,4	a	4 sampai 6	-6	-6	4	4
b	0,3	b	3 sampai 7	-7	3	-7	3
$a \vee b$	0,8	$\neg(a \vee b)$	2 sampai 8	2	2	2	-8
				-11	-1	-1	-1

Gambar 4.2 Karena Agen 1 memiliki keyakinan yang tidak konsisten, Agen 2 dapat merancang serangkaian taruhan yang menjamin kekalahan bagi Agen 1, tidak peduli apa pun hasil a dan b .

Satu keberatan umum terhadap teorema de Finetti adalah bahwa permainan taruhan ini agak dibuat-buat. Misalnya, bagaimana jika seseorang menolak untuk bertaruh? Apakah itu mengakhiri argumen? Jawabannya adalah bahwa permainan taruhan adalah model abstrak untuk situasi pengambilan keputusan di mana setiap agen terlibat secara tak terelakkan pada setiap momen. Setiap tindakan (termasuk tidak bertindak) adalah sejenis taruhan, dan setiap

hasil dapat dilihat sebagai hasil dari taruhan tersebut. Menolak untuk bertaruh seperti menolak untuk membiarkan waktu berlalu.

Argumen filosofis kuat lainnya telah diajukan untuk penggunaan probabilitas, terutama argumen Cox, Carnap, dan Jaynes. Masing-masing dari mereka membangun serangkaian aksioma untuk penalaran dengan tingkat keyakinan: tidak ada kontradiksi, korespondensi dengan logika biasa (misalnya, jika keyakinan pada A meningkat, maka keyakinan pada $\neg A$ pasti menurun), dan seterusnya. Satu-satunya aksioma yang kontroversial adalah bahwa derajat keyakinan harus berupa angka, atau setidaknya bertindak seperti angka karena harus bersifat transitif (jika keyakinan pada A lebih besar daripada keyakinan pada B , yang lebih besar daripada keyakinan pada C , maka keyakinan pada A harus lebih besar daripada C) dan sebanding (keyakinan pada A harus sama dengan, lebih besar daripada, atau kurang dari keyakinan pada B). Maka dapat dibuktikan bahwa probabilitas adalah satu-satunya pendekatan yang memenuhi aksioma ini. Namun, karena dunia seperti ini, demonstrasi praktis terkadang berbicara lebih keras daripada bukti. Keberhasilan sistem penalaran berdasarkan teori probabilitas jauh lebih efektif dalam membuat orang bertobat. Sekarang kita akan melihat bagaimana aksioma dapat digunakan untuk membuat kesimpulan.

4.3 INFERENSI MENGGUNAKAN DISTRIBUSI JOINT LENGKAP

Pada bagian ini kami menjelaskan metode sederhana untuk inferensi probabilistik yaitu, perhitungan probabilitas posterior untuk proposisi kueri berdasarkan bukti yang diamati. Kami menggunakan distribusi joint penuh sebagai "basis pengetahuan" yang dapat digunakan untuk memperoleh jawaban atas semua pertanyaan. Sepanjang jalan, kami juga memperkenalkan beberapa teknik yang berguna untuk memanipulasi persamaan yang melibatkan probabilitas.

DARI MANA ASALNYA KEMUNGKINAN?

Telah terjadi perdebatan tanpa akhir mengenai sumber dan status angka probabilitas. Posisi frekuentis adalah bahwa angka-angka tersebut hanya dapat berasal dari eksperimen: jika kita menguji 100 orang dan menemukan bahwa 10 dari mereka memiliki cavity, maka kita dapat mengatakan bahwa probabilitas cavity adalah sekitar 0,1. Dalam pandangan ini, pernyataan "probabilitas cavity adalah 0,1" berarti bahwa 0,1 adalah pecahan yang akan diamati dalam batas sampel yang tak terhingga banyaknya. Dari sampel yang terbatas, kita dapat memperkirakan pecahan sebenarnya dan juga menghitung seberapa akurat perkiraan kita.

Pandangan objektif adalah bahwa probabilitas adalah aspek nyata dari alam semesta—kecenderungan objek untuk berperilaku dengan cara tertentu—daripada hanya deskripsi tingkat keyakinan pengamat. Misalnya, fakta bahwa koin yang adil muncul dengan probabilitas 0,5 adalah kecenderungan koin itu sendiri. Dalam pandangan ini, pengukuran frekuentis merupakan upaya untuk mengamati kecenderungan-kecenderungan ini. Sebagian besar fisikawan setuju bahwa fenomena kuantum bersifat probabilistik secara objektif, tetapi ketidakpastian pada skala makroskopis—misalnya, dalam melempar koin—biasanya muncul

dari ketidaktahuan akan kondisi awal dan tampaknya tidak konsisten dengan pandangan kecenderungan.

Pandangan subjektif menggambarkan probabilitas sebagai cara untuk mengkarakterisasi keyakinan agen, daripada memiliki signifikansi fisik eksternal apa pun. Pandangan Bayesian subjektif memungkinkan setiap atribusi yang konsisten dengan sendirinya dari probabilitas sebelumnya terhadap proposisi, tetapi kemudian bersikeras pada pembaruan Bayesian yang tepat saat bukti tiba.

Pada akhirnya, bahkan posisi frequentist yang ketat melibatkan analisis subjektif karena masalah kelas referensi: dalam mencoba menentukan probabilitas hasil dari suatu eksperimen tertentu, frequentist harus menempatkannya dalam kelas referensi dari eksperimen "serupa" dengan frekuensi hasil yang diketahui. I. J. Good menulis, "setiap peristiwa dalam hidup adalah unik, dan setiap probabilitas kehidupan nyata yang kita perkirakan dalam praktik adalah probabilitas dari peristiwa yang belum pernah terjadi sebelumnya." Misalnya, diberikan pasien tertentu, seorang frequentist yang ingin memperkirakan probabilitas cavity akan mempertimbangkan kelas referensi pasien lain yang serupa dalam cara-cara penting—usia, gejala, diet—dan melihat berapa proporsi dari mereka yang memiliki cavity. Jika dokter tooth mempertimbangkan segala sesuatu yang diketahui tentang pasien—berat hingga gram terdekat, warna rambut, nama gadis ibu—maka kelas referensinya menjadi kosong. Ini telah menjadi masalah yang menjengkelkan dalam filsafat sains.

Prinsip ketidakpedulian yang dikaitkan dengan Laplace menyatakan bahwa proposisi yang secara sintaksis "simetris" berkenaan dengan bukti harus diberi probabilitas yang sama. Berbagai penyempurnaan telah diajukan, yang berpuncak pada upaya Carnap dan yang lainnya untuk mengembangkan logika induktif yang ketat, yang mampu menghitung probabilitas yang benar untuk setiap proposisi dari kumpulan pengamatan apa pun. Saat ini, diyakini bahwa tidak ada logika induktif yang unik; sebaliknya, logika semacam itu bergantung pada distribusi probabilitas subjektif sebelumnya yang efeknya berkurang seiring dengan semakin banyaknya pengamatan yang dikumpulkan.

	<i>toothache</i>		$\neg$ <i>toothache</i>	
	<i>catch</i>	$\neg$ <i>catch</i>	<i>catch</i>	$\neg$ <i>catch</i>
<i>cavity</i>	0.108	0.012	0.072	0.008
$\neg$ <i>cavity</i>	0.016	0.064	0.144	0.576

Gambar 4.3 Distribusi gabungan penuh untuk dunia *Toothache*, *Cavity*, dan *Catch*.

Kita mulai dengan contoh sederhana: domain yang hanya terdiri dari tiga variabel Boolean *Toothache*, *Cavity*, dan *Catch* (probe baja yang menyebalkan dari dokter tooth tersangkut di tooth saya). Distribusi gabungan lengkap adalah tabel $2 \times 2 \times 2$ seperti yang ditunjukkan pada Gambar 4.3.

Perhatikan bahwa probabilitas dalam distribusi gabungan berjumlah 1, seperti yang dipersyaratkan oleh aksioma probabilitas. Perhatikan juga bahwa Persamaan (4.2) memberi kita cara langsung untuk menghitung probabilitas proposisi apa pun, sederhana atau

kompleks: cukup identifikasi kemungkinan dunia di mana proposisi itu benar dan jumlahkan probabilitasnya. Misalnya, ada enam kemungkinan dunia di mana *cavity* $\vee$ *toothache* berlaku:

$$P(\text{cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28 .$$

Salah satu tugas yang sangat umum adalah mengekstrak distribusi atas beberapa subset variabel atau variabel tunggal. Misalnya, menambahkan entri di baris pertama memberikan probabilitas tanpa syarat atau marginal dari *cavity*:

$$P(\text{cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2 .$$

Proses ini disebut marginalisasi, atau penjumlahan keluar—karena kita menjumlahkan probabilitas untuk setiap nilai yang mungkin dari variabel-variabel lain, sehingga mengeluarkannya dari persamaan. Kita dapat menulis aturan marginalisasi umum berikut untuk setiap set variabel Y dan Z :

$$P(Y) = \sum_{z \in Z} P(Y, z) \quad (13.6)$$

di mana $\sum_{z \in Z}$ berarti menjumlahkan semua kemungkinan kombinasi nilai dari himpunan variabel Z . Kita terkadang meningkatkannya menjadi $\sum_z$, sehingga Z tersirat. Kita hanya menggunakan aturan sebagai

$$P(\text{Cavity}) = \sum_{z \in \{\text{Catch}, \text{Toothache}\}} P(\text{Cavity}, z) . \quad (13.7)$$

Varian dari aturan ini melibatkan probabilitas bersyarat, bukan probabilitas gabungan, dengan menggunakan aturan produk:

$$P(Y) = \sum_z P(Y | z) P(z) . \quad (13.8)$$

Aturan ini disebut pengkondisian. Marginalisasi dan pengkondisian ternyata menjadi aturan yang berguna untuk semua jenis derivasi yang melibatkan ekspresi probabilitas.

Dalam kebanyakan kasus, kita tertarik untuk menghitung probabilitas bersyarat dari beberapa variabel, dengan bukti tentang variabel lainnya. Probabilitas bersyarat dapat ditemukan dengan terlebih dahulu menggunakan Persamaan (4.3) untuk memperoleh ekspresi dalam bentuk probabilitas tanpa syarat dan kemudian mengevaluasi ekspresi tersebut dari distribusi gabungan penuh. Misalnya, kita dapat menghitung probabilitas *cavity*, dengan bukti *toothache*, sebagai berikut:

$$\begin{aligned}
 P(\text{cavity} | \text{toothache}) &= \frac{P(\text{cavity} | \text{toothache})}{P(\text{toothache})} \\
 &= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6.
 \end{aligned}$$

Untuk memeriksa, kita juga dapat menghitung probabilitas tidak adanya cavity, jika kita sedang toothache:

$$\begin{aligned}
 P(\neg \text{cavity} | \text{toothache}) &= \frac{P(\neg \text{cavity} \wedge \text{toothache})}{P(\text{toothache})} \\
 &= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4.
 \end{aligned}$$

Kedua nilai tersebut berjumlah 1,0, sebagaimana mestinya. Perhatikan bahwa dalam kedua perhitungan ini suku $1/P(\text{toothache})$ tetap konstan, tidak peduli nilai *Cavity* mana yang kita hitung. Bahkan, suku ini dapat dilihat sebagai konstanta normalisasi untuk distribusi $P(\text{Cavity} | \text{toothache})$, yang memastikan bahwa jumlahnya mencapai 1. Di seluruh bab yang membahas probabilitas, kita menggunakan α untuk menunjukkan konstanta tersebut. Dengan notasi ini, kita dapat menuliskan dua persamaan sebelumnya menjadi satu:

$$\begin{aligned}
 P(\text{Cavity} | \text{toothache}) &= \alpha P(\text{Cavity}, \text{toothache}) \\
 &= \alpha [P(\text{Cavity}, \text{toothache}, \text{catch}) + P(\text{Cavity}, \text{toothache}, \neg \text{catch})] \\
 &= \alpha [\langle 0.108, 0.016 \rangle + \langle 0.012, 0.064 \rangle] = \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle.
 \end{aligned}$$

Dengan kata lain, kita dapat menghitung $P(\text{Cavity} | \text{toothache})$ bahkan jika kita tidak mengetahui nilai $P(\text{toothache})$! Kita lupakan sejenak faktor $1/P(\text{toothache})$ dan jumlahkan nilai untuk *cavity* dan $\neg \text{cavity}$, sehingga memperoleh 0,12 dan 0,08. Itu adalah proporsi relatif yang benar, tetapi jumlahnya tidak mencapai 1, jadi kita normalkan dengan membagi masing-masing dengan $0,12 + 0,08$, sehingga memperoleh probabilitas sebenarnya 0,6 dan 0,4. Normalisasi ternyata menjadi jalan pintas yang berguna dalam banyak kalkulasi probabilitas, baik untuk mempermudah komputasi maupun untuk memungkinkan kita melanjutkan ketika beberapa penilaian probabilitas (seperti $P(\text{toothache})$) tidak tersedia.

Dari contoh tersebut, kita dapat mengekstrak prosedur inferensi umum. Kita mulai dengan kasus di mana kueri melibatkan satu variabel, $X(\text{Cavity}$ dalam contoh). Misalkan E adalah daftar variabel bukti (hanya *Toothache* dalam contoh), misalkan e adalah daftar nilai yang diamati untuk variabel tersebut, dan misalkan Y adalah variabel yang tidak diamati yang tersisa (hanya *Catch* dalam contoh). Kueri tersebut adalah $P(X|e)$ dan dapat dievaluasi sebagai

$$P(X | e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y), \quad (13.9)$$

di mana penjumlahannya adalah atas semua kemungkinan y_s (yaitu, semua kemungkinan kombinasi nilai variabel Y yang tidak teramati). Perhatikan bahwa bersama-sama variabel X, E , dan Y membentuk himpunan variabel lengkap untuk domain, jadi $P(X, e, y)$ hanyalah bagian dari probabilitas dari distribusi gabungan penuh.

Dengan distribusi gabungan penuh yang harus dikerjakan, Persamaan (4.9) dapat menjawab pertanyaan probabilistik untuk variabel diskrit. Namun, hal itu tidak berskala dengan baik: untuk domain yang dijelaskan oleh n variabel Boolean, diperlukan tabel input berukuran $O(2^n)$ dan memerlukan waktu $O(2^n)$ untuk memproses tabel tersebut. Dalam masalah yang realistis, kita dapat dengan mudah memiliki $n > 100$, sehingga $O(2^n)$ tidak praktis. Distribusi gabungan penuh dalam bentuk tabel bukanlah alat praktis untuk membangun sistem penalaran. Sebaliknya, hal itu harus dilihat sebagai landasan teoritis yang dapat digunakan untuk membangun pendekatan yang lebih efektif, seperti tabel kebenaran yang membentuk landasan teoritis untuk algoritma yang lebih praktis seperti DPLL.

4.4 KEMANDIRIAN

Mari kita perluas distribusi gabungan lengkap pada Gambar 4.3 dengan menambahkan variabel keempat, *Weather*. Distribusi gabungan lengkap kemudian menjadi $P(\textit{Toothache}, \textit{Catch}, \textit{Cavity}, \textit{Weather})$, yang memiliki $2 \times 2 \times 2 \times 4 = 32$ entri. Ini berisi empat "edisi" tabel yang ditunjukkan pada Gambar 4.3, satu untuk setiap jenis *Weather*. Apa hubungan edisi-edisi ini satu sama lain dan dengan tabel tiga variabel asli? Misalnya, bagaimana $P(\textit{toothache}, \textit{catch}, \textit{cavity}, \textit{cloudy})$ dan $P(\textit{toothache}, \textit{catch}, \textit{cavity})$ terkait? Kita dapat menggunakan aturan perkalian:

$$\begin{aligned} P(\textit{toothache}, \textit{catch}, \textit{cavity}, \textit{cloudy}) \\ = P(\textit{cloudy} | \textit{toothache}, \textit{catch}, \textit{cavity}) P(\textit{toothache}, \textit{catch}, \textit{cavity}) . \end{aligned}$$

Nah, kecuali jika seseorang berkecimpung dalam bisnis ketuhanan, seseorang tidak boleh membayangkan bahwa masalah *tooth* seseorang memengaruhi *Weather*. Dan untuk kedokteran *tooth* dalam ruangan, setidaknya, tampaknya aman untuk mengatakan bahwa *Weather* tidak memengaruhi variabel *tooth*. Oleh karena itu, pernyataan berikut tampaknya masuk akal:

$$P(\textit{cloudy} | \textit{toothache}, \textit{catch}, \textit{cavity}) = P(\textit{cloudy}) . \quad (13.10)$$

Dari sini kita dapat menyimpulkan

$$P(\textit{toothache}, \textit{catch}, \textit{cavity}) = P(\textit{cloudy}) P(\textit{toothache}, \textit{catch}, \textit{cavity}) .$$

Persamaan serupa berlaku untuk setiap entri di $P(\textit{Toothache}, \textit{Catch}, \textit{Cavity}, \textit{Weather})$. Faktanya, kita dapat menuliskan persamaan umum

$$P(\text{Toothache}, \text{Catch}, \text{Cavity}, \text{Weather}) \\ = P(\text{Toothache}, \text{Catch}, \text{Cavity})P(\text{Weather}) .$$

Dengan demikian, tabel 32 elemen untuk empat variabel dapat dibangun dari satu tabel 8 elemen dan satu tabel 4 elemen. Dekomposisi ini diilustrasikan secara skematis pada Gambar 4.4(a).

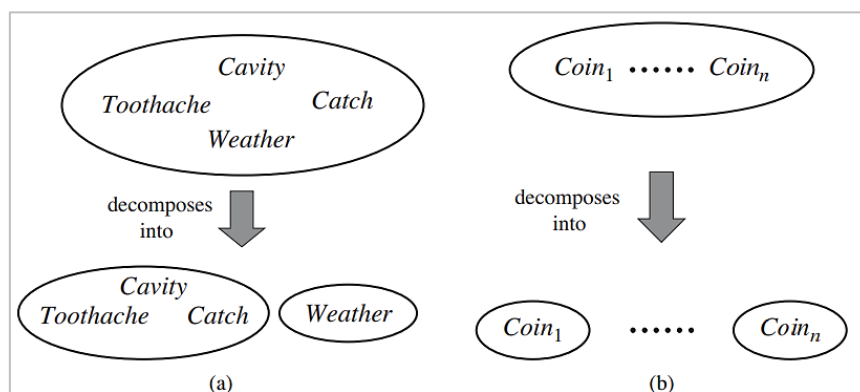
Properti yang kami gunakan dalam Persamaan (4.10) disebut independensi (juga disebut independensi marginal dan independensi absolut). Secara khusus, Weather tidak bergantung pada masalah tooth seseorang. Independensi antara proposisi a dan b dapat ditulis sebagai

$$P(a | b) = P(a) \text{ or } P(b | a) = P(b) \text{ or } P(a \wedge b) = P(a)P(b) . \quad (13.11)$$

Semua bentuk ini ekuivalen. Kemandirian antara variabel X dan Y dapat ditulis sebagai berikut:

$$P(X | Y) = P(X) \text{ or } P(Y | X) = P(Y) \text{ or } P(X, Y) = P(X)P(Y) .$$

Pernyataan independensi biasanya didasarkan pada pengetahuan tentang domain. Seperti yang diilustrasikan dalam contoh toothache–weather, pernyataan independensi dapat secara drastis mengurangi jumlah informasi yang diperlukan untuk menentukan distribusi gabungan penuh. Jika rangkaian variabel lengkap dapat dibagi menjadi beberapa bagian yang independen, maka distribusi gabungan penuh dapat difaktorkan menjadi distribusi gabungan terpisah pada bagian-bagian tersebut.



Gambar 4.4 Dua contoh pemfaktoran distribusi gabungan yang besar ke dalam distribusi yang lebih kecil, menggunakan independensi absolut. (a) Masalah Weather dan tooth bersifat independen. (b) Lemparan koin bersifat independen.

Misalnya, distribusi gabungan penuh pada hasil n lemparan koin yang independen, $P(C_1, \dots, C_n)$, memiliki 2^n entri, tetapi dapat direpresentasikan sebagai produk dari n

distribusi variabel tunggal $P(C_1)$. Dalam konteks yang lebih praktis, independensi kedokteran tooth dan meteorologi merupakan hal yang baik, karena jika tidak, praktik kedokteran tooth mungkin memerlukan pengetahuan mendalam tentang meteorologi, dan sebaliknya.

Jika tersedia, pernyataan independensi dapat membantu mengurangi ukuran representasi domain dan kompleksitas masalah inferensi. Sayangnya, pemisahan yang jelas dari seluruh rangkaian variabel dengan independensi cukup jarang. Setiap kali ada hubungan, betapapun tidak langsungnya, antara dua variabel, independensi akan gagal berlaku. Terlebih lagi, bahkan subset yang independen pun bisa sangat besar—misalnya, kedokteran tooth mungkin melibatkan puluhan penyakit dan ratusan gejala, yang semuanya saling terkait. Untuk menangani masalah seperti itu, kita memerlukan metode yang lebih halus daripada konsep independensi yang lugas.

4.5 ATURAN BAYES DAN PENGGUNAANNYA

Kami mendefinisikan aturan perkalian. Aturan ini sebenarnya dapat ditulis dalam dua bentuk:

$$P(a \wedge b) = P(a | b)P(b) \text{ dan } P(a \wedge b) = P(b | a)P(a).$$

Dengan menyamakan kedua ruas kanan dan membaginya dengan $P(a)$, kita peroleh

$$P(b | a) = \frac{P(a | b)P(b)}{P(a)}. \quad (13.12)$$

Persamaan ini dikenal sebagai aturan Bayes (juga hukum Bayes atau teorema Bayes). Persamaan sederhana ini mendasari sebagian besar sistem AI modern untuk inferensi probabilistik.

Kasus aturan Bayes yang lebih umum untuk variabel bernilai banyak dapat ditulis dalam notasi P sebagai berikut:

$$P(Y | X) = \frac{P(X | Y)P(Y)}{P(X)}$$

Seperti sebelumnya, ini harus dianggap mewakili serangkaian persamaan, yang masing-masing membahas nilai variabel tertentu. Kita juga akan memiliki kesempatan untuk menggunakan versi yang lebih umum dengan syarat beberapa bukti latar belakang e :

$$P(Y | X, e) = \frac{P(X | Y, e)P(Y | e)}{P(X | e)} \quad (13.13)$$

Menerapkan Aturan Bayes: Kasus Sederhana

Di permukaan, aturan Bayes tampaknya tidak terlalu berguna. Aturan ini memungkinkan kita menghitung suku tunggal $P(b | a)$ dalam tiga suku: $P(a | b)$, $P(b)$, dan $P(a)$. Itu tampak seperti dua langkah mundur, tetapi aturan Bayes berguna dalam praktik karena ada banyak kasus di mana kita memiliki estimasi probabilitas yang baik untuk ketiga angka ini dan perlu menghitung angka keempat. Sering kali, kita melihat sebagai bukti pengaruh dari beberapa penyebab yang tidak diketahui dan kita ingin menentukan penyebab itu. Dalam kasus itu, aturan Bayes menjadi

$$P(\text{cause} | \text{effect}) = \frac{P(\text{effect} | \text{cause})P(\text{cause})}{P(\text{effect})}.$$

Probabilitas bersyarat $P(\text{cause} | \text{effect})$ mengukur hubungan dalam arah kausal, sedangkan $P(\text{cause} | \text{effect})$ menggambarkan arah diagnostik. Dalam tugas seperti diagnosis medis, kita sering memiliki probabilitas bersyarat pada hubungan kausal (yaitu, dokter mengetahui $P(\text{symptoms} | \text{disease})$ dan ingin memperoleh diagnosis, $P(\text{disease} | \text{symptoms})$. Misalnya, seorang dokter mengetahui bahwa penyakit meningitis menyebabkan pasien mengalami leher kaku, katakanlah, 70% dari waktu. Dokter juga mengetahui beberapa fakta tanpa syarat: probabilitas sebelumnya bahwa seorang pasien mengalami meningitis adalah 1/50.000, dan probabilitas sebelumnya bahwa setiap pasien mengalami leher kaku adalah 1%. Misalkan s adalah proposisi bahwa pasien mengalami leher kaku dan m adalah proposisi bahwa pasien mengalami meningitis, kita memperoleh

$$\begin{aligned} P(s | m) &= 0.7 \\ P(m) &= 1/50000 \\ P(s) &= 0.01 \end{aligned} \tag{13.14}$$

$$P(m | s) = \frac{P(s | m)P(m)}{P(s)} = \frac{0.7 \times 1/50000}{0.01} = 0.0014.$$

Artinya, kami memperkirakan kurang dari 1 dari 700 pasien dengan leher kaku mengalami meningitis. Perhatikan bahwa meskipun leher kaku cukup kuat diindikasikan oleh meningitis (dengan probabilitas 0,7), probabilitas meningitis pada pasien tetap kecil. Ini karena probabilitas leher kaku sebelumnya jauh lebih tinggi daripada meningitis.

Bagian selanjutnya mengilustrasikan proses yang dengannya seseorang dapat menghindari penilaian probabilitas sebelumnya dari bukti (di sini, $P(s)$) dengan menghitung probabilitas posterior untuk setiap nilai variabel kueri (di sini, m dan $\neg m$) dan kemudian menormalkan hasilnya. Proses yang sama dapat diterapkan saat menggunakan aturan Bayes. Kami memiliki

$$P(M | s) = \alpha \langle P(s | m)P(m), P(s | \neg m)P(\neg m) \rangle.$$

Jadi, untuk menggunakan pendekatan ini kita perlu memperkirakan $P(s | \neg m)$ dan bukan $P(s)$. Tidak ada makan siang gratis—kadang-kadang ini lebih mudah, kadang-kadang lebih sulit. Bentuk umum aturan Bayes dengan normalisasi adalah

$$P(Y | X) = \alpha P(X | Y)P(Y) , \quad (13.15)$$

di mana α adalah konstanta normalisasi yang dibutuhkan untuk membuat entri dalam $P(Y | X)$ berjumlah 1.

Satu pertanyaan yang jelas untuk ditanyakan tentang aturan Bayes adalah mengapa seseorang mungkin memiliki probabilitas bersyarat yang tersedia dalam satu arah, tetapi tidak pada arah yang lain. Dalam domain meningitis, mungkin dokter mengetahui bahwa leher kaku menyiratkan meningitis dalam 1 dari 5000 kasus; yaitu, dokter memiliki informasi kuantitatif dalam arah diagnostik dari gejala ke penyebab. Dokter seperti itu tidak perlu menggunakan aturan Bayes. Sayangnya, pengetahuan diagnostik seringkali lebih rapuh daripada pengetahuan kausal. Jika ada epidemi meningitis yang tiba-tiba, probabilitas meningitis tanpa syarat, $P(m)$, akan naik. Dokter yang memperoleh probabilitas diagnostik $P(m | s)$ secara langsung dari pengamatan statistik pasien sebelum epidemi tidak akan tahu bagaimana memperbarui nilai tersebut, tetapi dokter yang menghitung $P(m | s)$ dari tiga nilai lainnya akan melihat bahwa $P(m | s)$ harus naik secara proporsional dengan $P(m)$. Yang terpenting, informasi kausal $P(s | m)$ tidak terpengaruh oleh epidemi, karena hanya mencerminkan cara kerja meningitis. Penggunaan pengetahuan kausal langsung atau berbasis model semacam ini memberikan ketahanan penting yang dibutuhkan untuk membuat sistem probabilistik layak di dunia nyata.

Menggunakan Aturan Bayes: Menggabungkan Bukti

Kita telah melihat bahwa aturan Bayes dapat berguna untuk menjawab pertanyaan probabilistik yang dikondisikan pada satu bukti—misalnya, leher kaku. Secara khusus, kami berpendapat bahwa informasi probabilistik sering kali tersedia dalam bentuk $P(\text{effect} | \text{cause})$. Apa yang terjadi ketika kita memiliki dua atau lebih bukti? Misalnya, apa yang dapat disimpulkan oleh seorang dokter gigi jika probe baja yang tidak enak mengenai gigi pasien yang sakit? Jika kita mengetahui distribusi sendi secara lengkap (Gambar 4.3), kita dapat membaca jawabannya:

$$P(\text{Cavity} | \text{toothache} \wedge \text{catch}) = \alpha \langle 0.108, 0.016 \rangle \approx \langle 0.871, 0.129 \rangle .$$

Namun, kita tahu bahwa pendekatan semacam itu tidak dapat diterapkan pada jumlah variabel yang lebih besar. Kita dapat mencoba menggunakan aturan Bayes untuk merumuskan kembali masalah tersebut:

$$\begin{aligned} P(\text{Cavity} | \text{toothache} \wedge \text{catch}) \\ = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity})P(\text{Cavity}) . \end{aligned} \quad (13.16)$$

Agar rumusan ulang ini berhasil, kita perlu mengetahui probabilitas kondisional dari konjungsi *toothache* $\wedge$ *catch* untuk setiap nilai *Cavity*. Itu mungkin layak untuk hanya dua variabel bukti, tetapi sekali lagi itu tidak dapat ditingkatkan. Jika ada n variabel bukti yang mungkin (sinar X, diet, kebersihan mulut, dll.), maka ada 2^n kemungkinan kombinasi nilai yang diamati yang untuknya kita perlu mengetahui probabilitas kondisional. Kita mungkin juga kembali menggunakan distribusi gabungan penuh. Inilah yang pertama kali mengarahkan para peneliti menjauh dari teori probabilitas menuju metode perkiraan untuk kombinasi bukti yang, meskipun memberikan jawaban yang salah, memerlukan lebih sedikit angka untuk memberikan jawaban apa pun.

Alih-alih mengambil rute ini, kita perlu menemukan beberapa pernyataan tambahan mengenai domain yang akan memungkinkan kita menyederhanakan ekspresi-ekspresi tersebut. Gagasan independensi di Bagian 4.4 memberikan petunjuk, tetapi perlu penyempurnaan. Akan menyenangkan jika *Toothache* dan *Catch* itu independen, tetapi tidak: jika probe menangkap di gigi, maka kemungkinan besar gigi tersebut memiliki karies (lubang) dan karies tersebut menyebabkan sakit gigi. Namun, variabel-variabel ini independen, dengan adanya atau tidak adanya karies (lubang gigi). Masing-masing secara langsung disebabkan oleh karies, tetapi tidak ada yang memiliki efek langsung pada yang lain: *Toothache* tergantung pada keadaan saraf di gigi, sedangkan akurasi probe tergantung pada keterampilan dokter gigi, yang mana *toothache* tidak relevan dengannya. Secara matematis, sifat ini ditulis sebagai

$$P(\text{toothache} \wedge \text{catch} \mid \text{Cavity}) = P(\text{toothache} \mid \text{Cavity})P(\text{catch} \mid \text{Cavity}). \quad (4.17)$$

Persamaan ini menyatakan independensi bersyarat dari *toothache* dan *catch* berdasarkan *Cavity*. Kita dapat memasukkannya ke dalam Persamaan (4.16) untuk memperoleh probabilitas terjadinya cavity:

$$\begin{aligned} P(\text{Cavity} \mid \text{toothache} \wedge \text{catch}) \\ = \alpha P(\text{toothache} \mid \text{Cavity})P(\text{catch} \mid \text{Cavity})P(\text{Cavity}). \end{aligned} \quad (4.18)$$

Sekarang persyaratan informasi sama seperti untuk inferensi, menggunakan setiap bagian bukti secara terpisah: probabilitas sebelumnya $P(\text{Cavity})$ untuk variabel kueri dan probabilitas bersyarat dari setiap efek, dengan mempertimbangkan penyebabnya.

Definisi umum independensi bersyarat dari dua variabel X dan Y , dengan mempertimbangkan variabel ketiga Z , adalah

$$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z).$$

Dalam domain dokter tooth, misalnya, tampaknya masuk akal untuk menegaskan independensi bersyarat dari variabel *Toothache* dan *Catch*, mengingat *Cavity* :

$$P(\text{Toothache}, \text{Catch} \mid \text{Cavity}) = P(\text{Toothache} \mid \text{Cavity})P(\text{Catch} \mid \text{Cavity}). \quad (4.19)$$

Perhatikan bahwa pernyataan ini agak lebih kuat daripada Persamaan (4.17), yang menegaskan independensi hanya untuk nilai-nilai tertentu dari *Toothache* dan *Catch*. Seperti halnya independensi absolut dalam Persamaan (4.11), bentuk-bentuk yang setara

$$P(X \mid Y, Z) = P(X \mid Z) \text{ dan } P(Y \mid X, Z) = P(Y \mid Z)$$

juga dapat digunakan. Pernyataan independensi absolut memungkinkan dekomposisi distribusi gabungan penuh menjadi bagian-bagian yang jauh lebih kecil. Ternyata hal yang sama berlaku untuk pernyataan independensi bersyarat. Misalnya, dengan pernyataan dalam Persamaan (4.19), kita dapat memperoleh dekomposisi sebagai berikut:

$$\begin{aligned} P(\text{Toothache}, \text{Catch}, \text{Cavity}) \\ &= P(\text{Toothache}, \text{Catch} \mid \text{Cavity})P(\text{Cavity}) \quad (\text{product rule}) \\ &= P(\text{Toothache} \mid \text{Cavity})P(\text{Catch} \mid \text{Cavity})P(\text{Cavity}) \quad (\text{using 13.19}) \end{aligned}$$

Dengan cara ini, tabel besar asli diurai menjadi tiga tabel yang lebih kecil. Tabel asli memiliki tujuh angka independen ($2^3 = 8$ entri dalam tabel, tetapi jumlahnya harus 1, jadi 7 adalah independen). Tabel yang lebih kecil berisi lima angka independen (untuk distribusi probabilitas bersyarat seperti $P(T \mid C)$ ada dua baris yang terdiri dari dua angka, dan setiap baris berjumlah 1, jadi itu adalah dua angka independen; untuk distribusi prior seperti $P(C)$ hanya ada satu angka independen). Beralih dari tujuh ke lima mungkin tidak tampak seperti kemenangan besar, tetapi intinya adalah, untuk n gejala yang semuanya independen secara bersyarat yang diberikan *Cavity*, ukuran representasi tumbuh sebagai $O(n)$ alih-alih $O(2^n)$. Itu berarti bahwa pernyataan independensi bersyarat dapat memungkinkan sistem probabilistik untuk ditingkatkan; lebih jauh lagi, mereka jauh lebih umum tersedia daripada pernyataan independensi absolut. Secara konseptual, *Cavity* memisahkan *Toothache* dan *Catch* karena merupakan penyebab langsung dari keduanya. Dekomposisi domain probabilistik besar menjadi subset yang terhubung lemah melalui independensi bersyarat adalah salah satu perkembangan terpenting dalam sejarah AI terkini.

Contoh kedokteran tooth menggambarkan pola yang umum terjadi di mana satu penyebab secara langsung memengaruhi sejumlah efek, semuanya bersifat independen secara kondisional, mengingat penyebabnya. Distribusi gabungan lengkap dapat ditulis sebagai

$$P(\text{Cause}, \text{Effect}_1, \dots, \text{Effect}_n) = P(\text{Cause}) \prod_i P(\text{Effect}_i \mid \text{Cause})$$

Distribusi probabilitas seperti itu disebut model Bayes naif—“naif” karena sering digunakan (sebagai asumsi penyederhanaan) dalam kasus di mana variabel “akibat” sebenarnya tidak

independen secara kondisional mengingat variabel penyebab. (Model Bayes naif terkadang disebut pengklasifikasi Bayesian, penggunaan yang agak ceroboh yang telah mendorong penganut Bayes sejati untuk menyebutnya model idiot Bayes.) Dalam praktiknya, sistem Bayes naif dapat bekerja dengan sangat baik, bahkan ketika asumsi independensi kondisional tidak benar.

4.6 DUNIA WUMPUS DIKUNJUNGI KEMBALI

Kita dapat menggabungkan ide-ide dalam bab ini untuk memecahkan masalah penalaran probabilistik di dunia wumpus. Ketidakpastian muncul di dunia wumpus karena sensor agen hanya memberikan informasi parsial tentang dunia tersebut. Misalnya, Gambar 4.5 menunjukkan situasi di mana masing-masing dari tiga kotak yang dapat dijangkau—[1,3], [2,2], dan [3,1]—mungkin berisi sebuah lubang. Inferensi logis murni tidak dapat menyimpulkan apa pun tentang kotak mana yang kemungkinan besar aman, jadi agen logis mungkin harus memilih secara acak. Kita akan melihat bahwa agen probabilistik dapat melakukannya jauh lebih baik daripada agen logis.

Tujuan kita adalah menghitung probabilitas bahwa masing-masing dari tiga kotak berisi sebuah lubang. (Untuk contoh ini kita mengabaikan wumpus dan emas.) Properti yang relevan dari dunia wumpus adalah bahwa (1) sebuah lubang menyebabkan angin sepoi-sepoi di semua kotak di sekitarnya, dan (2) setiap kotak selain [1,1] berisi sebuah lubang dengan probabilitas 0,2. Langkah pertama adalah mengidentifikasi kumpulan variabel acak yang kita butuhkan:

- Seperti dalam kasus logika proposisional, kita menginginkan satu variabel Boolean P_{ij} untuk setiap kotak, yang benar jika kotak $[i, j]$ benar-benar berisi sebuah lubang.



- Kita juga memiliki variabel Boolean B_{ij} yang bernilai benar jika kuadrat $[i, j]$ adalah breezy; kita hanya menyertakan variabel ini untuk kuadrat yang diamati—dalam kasus ini, [1,1], [1,2], dan [2,1].

Langkah berikutnya adalah menentukan distribusi gabungan penuh, $P(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$. Dengan menerapkan aturan produk, kita memiliki

$$P(P_{1,1}, \dots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = \\ P(B_{1,1}, B_{1,2}, B_{2,1} \mid P_{1,1}, \dots, P_{4,4})P(P_{1,1}, \dots, P_{4,4})$$

Dekomposisi ini memudahkan untuk melihat berapa nilai probabilitas gabungan yang seharusnya. Suku pertama adalah distribusi probabilitas bersyarat dari konfigurasi angin sepoi-sepoi, jika diberikan konfigurasi lubang; nilainya adalah 1 jika angin sepoi-sepoi berdekatan dengan lubang dan 0 jika tidak. Suku kedua adalah probabilitas sebelumnya dari konfigurasi lubang. Setiap kotak berisi lubang dengan probabilitas 0,2, terlepas dari kotak lainnya; oleh karena itu,

$$P(P_{1,1}, \dots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} P(P_{i,j}). \quad (4.20)$$

Untuk konfigurasi tertentu dengan tepat n lubang, $P(P_{1,1}, \dots, P_{4,4}) = 0,2^n \times 0,816^{-n}$.

Dalam situasi pada Gambar 4.5(a), buktinya terdiri dari angin sepoi-sepoi yang diamati (atau ketiadaannya) di setiap kotak yang dikunjungi, dikombinasikan dengan fakta bahwa setiap kotak tersebut tidak mengandung lubang. Kami menyingkat fakta-fakta ini sebagai $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ dan $known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$. Kami tertarik untuk menjawab pertanyaan seperti $P(P_{1,3} \mid known, b)$: seberapa besar kemungkinan [1,3] mengandung lubang, mengingat pengamatan sejauh ini?

Untuk menjawab pertanyaan ini, kita dapat mengikuti pendekatan standar Persamaan (4.9), yaitu, menjumlahkan entri dari distribusi gabungan penuh. Biarkan *Unknown* menjadi himpunan variabel $P_{i,j}$ untuk kuadrat selain kuadrat *Known* dan kuadrat query [1,3]. Maka, dengan Persamaan (4.9), kita peroleh

$$P(P_{1,3} \mid known, b) = \alpha \sum_{unknown} P(P_{P_{1,3}}, unknown, known, b).$$

Probabilitas gabungan penuh telah ditentukan, jadi kita sudah selesai—kecuali jika kita peduli dengan komputasi. Ada 12 kotak yang tidak diketahui; maka penjumlahan berisi $2^{12} = 4096$ suku. Secara umum, penjumlahan tumbuh secara eksponensial dengan jumlah kotak.

Tentunya, orang mungkin bertanya, bukankah kotak lainnya tidak relevan? Bagaimana [4,4] dapat memengaruhi apakah [1,3] memiliki lubang? Memang, intuisi ini benar. Biarkan *Frontier* menjadi variabel lubang (selain variabel kueri) yang berdekatan dengan kotak yang dikunjungi, dalam hal ini hanya [2,2] dan [3,1]. Juga, biarkan *Other* menjadi variabel lubang untuk kotak tidak diketahui lainnya; dalam hal ini, ada 10 kotak lainnya, seperti yang ditunjukkan pada Gambar 4.5(b). Wawasan utama adalah bahwa angin sepoi-sepoi yang diamati secara *kondisional independen* dari variabel lain, mengingat variabel yang diketahui, frontier, dan kueri. Untuk menggunakan wawasan ini, kita memanipulasi rumus kueri ke dalam

bentuk yang mana angin sepoi-sepoi dikondisikan pada semua variabel lainnya, dan kemudian kita menerapkan independensi bersyarat:

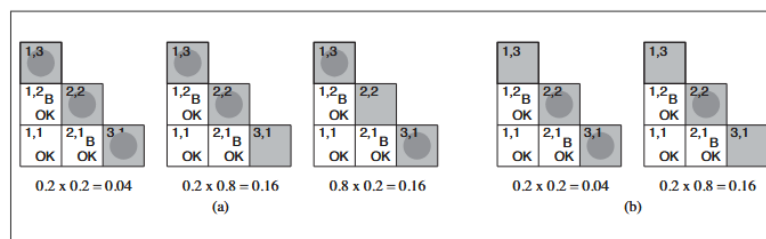
$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}) \\
 &= \alpha \sum_{\text{unknown}} \mathbf{P}(P_{1,3}, \text{known}, b, \text{unknown}) \quad (\text{by Equation (13.9)}) \\
 &= \alpha \sum_{\text{unknown}} \mathbf{P}(b | P_{1,3}, \text{known}, \text{unknown}) \mathbf{P}(P_{1,3}, \text{known}, \text{unknown}) \\
 & \quad \quad \quad (\text{Berdasarkan kaidah perkalian}) \\
 &= a \sum_{\text{frontier}} \sum_{\text{other}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}, \text{other}) \mathbf{P}(P_{1,3}, \text{known}, \text{frontier}, \text{other}) \\
 &= a \sum_{\text{frontier}} \sum_{\text{other}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}) \mathbf{P}(P_{1,3}, \text{known}, \text{frontier}, \text{other})
 \end{aligned}$$

di mana langkah terakhir menggunakan independensi bersyarat: b adalah independen dan tidak bergantung pada *variabel lain yang diketahui*, $P_{1,3}$, dan frontier . Sekarang, suku pertama dalam ekspresi ini tidak bergantung pada *variabel lain*, jadi kita dapat memindahkan penjumlahan ke dalam:

$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}, b) \\
 &= a \sum_{\text{frontier}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}) \sum_{\text{other}} \mathbf{P}(P_{1,3}, \text{known}, \text{frontier}, \text{other}).
 \end{aligned}$$

Dengan independensi, seperti pada Persamaan (4.20), suku sebelumnya dapat difaktorkan, dan kemudian suku-suku tersebut dapat disusun kembali:

$$\begin{aligned}
 & \mathbf{P}(P_{1,3} | \text{known}, b) \\
 &= a \sum_{\text{frontier}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}) \sum_{\text{other}} \mathbf{P}(P_{1,3}) \mathbf{P}(\text{known}) \mathbf{P}(\text{frontier}) \mathbf{P}(\text{other}) \\
 &= a' \mathbf{P}(\text{known}) \mathbf{P}(P_{1,3}) \sum_{\text{frontier}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}) \mathbf{P}(\text{frontier}), \sum_{\text{other}} \mathbf{P}(\text{other}) \\
 &= a' \mathbf{P}(P_{1,3}) \sum_{\text{frontier}} \mathbf{P}(b | \text{known}, P_{1,3}, \text{frontier}) \mathbf{P}(\text{frontier}),
 \end{aligned}$$



Gambar 4.6 Model konsisten untuk variabel frontier $P_{2,2}$ dan $P_{3,1}$, menunjukkan $P(\text{frontier})$ untuk setiap model: (a) tiga model dengan $P_{1,3} = \text{benar}$ yang menunjukkan

dua atau tiga lubang, dan (b) dua model dengan $P_{1,3}$ = salah yang menunjukkan satu atau dua lubang.

di mana langkah terakhir melipat $P(\text{known})$ ke dalam konstanta normalisasi dan menggunakan fakta bahwa $\sum_{\text{other}} P(\text{other})$ sama dengan 1.

Sekarang, hanya ada empat suku dalam penjumlahan atas variabel batas $P_{2,2}$ dan $P_{3,1}$. Penggunaan independensi dan independensi bersyarat telah sepenuhnya menghilangkan kuadrat lainnya dari pertimbangan.

Perhatikan bahwa ekspresi $P(b|\text{known}, P_{1,3}, \text{frontier})$ adalah 1 ketika batas konsisten dengan pengamatan angin, dan 0 sebaliknya. Jadi, untuk setiap nilai $P_{1,3}$, kami menjumlahkan *model logis* untuk variabel batas yang konsisten dengan fakta yang diketahui. (Bandingkan dengan enumerasi atas model pada Gambar 7.5 di halaman 241.) Model dan probabilitas sebelumnya yang terkait— $P(\text{frontier})$ —ditunjukkan pada Gambar 4.6. Kami memiliki

$$P(P_{1,3} | \text{known}, b) = \alpha' \langle 0.2(0.04 + 0.16 + 0.16), 0.8(0.04 + 0.16) \rangle \approx \langle 0.31, 0.69 \rangle.$$

Artinya, [1,3] (dan [3,1] berdasarkan simetri) berisi lubang dengan probabilitas sekitar 31%. Perhitungan serupa, yang mungkin ingin dilakukan pembaca, menunjukkan bahwa [2,2] berisi lubang dengan probabilitas sekitar 86%. Agen wumpus harus menghindari [2,2]! Perhatikan bahwa agen logis kita dari Bab 7 tidak tahu bahwa [2,2] lebih buruk daripada kotak lainnya. Logika dapat memberi tahu kita bahwa tidak diketahui apakah ada lubang di [2, 2], tetapi kita memerlukan probabilitas untuk memberi tahu kita seberapa besar kemungkinannya.

Apa yang ditunjukkan bagian ini adalah bahwa bahkan masalah yang tampaknya rumit dapat diformulasikan secara tepat dalam teori probabilitas dan dipecahkan dengan algoritme sederhana. Untuk mendapatkan solusi yang *efisien*, hubungan independensi dan independensi bersyarat dapat digunakan untuk menyederhanakan penjumlahan yang diperlukan. Hubungan ini sering kali sesuai dengan pemahaman alami kita tentang bagaimana masalah tersebut harus didekomposisi. Dalam bab berikutnya, kita mengembangkan representasi formal untuk hubungan tersebut serta algoritme yang beroperasi pada representasi tersebut untuk melakukan inferensi probabilistik secara efisien.

4.7 RINGKASAN

Bab ini telah menyarankan teori probabilitas sebagai landasan yang sesuai untuk penalaran yang tidak pasti dan memberikan pengantar yang lembut untuk penggunaannya.

- Ketidakpastian muncul karena kemalasan dan ketidaktahuan. Ketidakpastian tidak dapat dihindari dalam lingkungan yang kompleks, nondeterministik, atau yang dapat diamati sebagian.
- Probabilitas mengungkapkan ketidakmampuan agen untuk mencapai keputusan yang pasti mengenai kebenaran suatu kalimat. Probabilitas merangkum keyakinan agen yang relatif terhadap bukti.

- Teori keputusan menggabungkan keyakinan dan keinginan agen, mendefinisikan tindakan terbaik sebagai tindakan yang memaksimalkan utilitas yang diharapkan.
- Pernyataan probabilitas dasar mencakup **probabilitas prior** dan **probabilitas kondisional** atas proposisi sederhana dan kompleks.
- Aksioma probabilitas membatasi kemungkinan penugasan probabilitas pada proposisi. Agen yang melanggar aksioma harus berperilaku tidak rasional dalam beberapa kasus.
- Distribusi **probabilitas gabungan penuh** menentukan probabilitas setiap penugasan nilai lengkap ke variabel acak. Biasanya terlalu besar untuk dibuat atau digunakan dalam bentuk eksplisitnya, tetapi jika tersedia, dapat digunakan untuk menjawab pertanyaan hanya dengan menambahkan entri untuk kemungkinan dunia yang sesuai dengan proposisi pertanyaan.
- **Kemandirian absolut** antara subset variabel acak memungkinkan distribusi gabungan penuh untuk difaktorkan ke dalam distribusi gabungan yang lebih kecil, sehingga sangat mengurangi kompleksitasnya. Kemandirian absolut jarang terjadi dalam praktik.
- **Aturan Bayes** memungkinkan probabilitas yang tidak diketahui untuk dihitung dari probabilitas kondisional yang diketahui, biasanya dalam arah kausal. Menerapkan aturan Bayes dengan banyak bukti akan menemui masalah penskalaan yang sama seperti halnya distribusi gabungan penuh.
- **Kemandirian kondisional** yang ditimbulkan oleh hubungan kausal langsung dalam domain mungkin memungkinkan distribusi gabungan penuh untuk difaktorkan ke dalam distribusi kondisional yang lebih kecil. Model **Bayes naif** mengasumsikan independensi bersyarat dari semua variabel efek, dengan satu variabel penyebab, dan tumbuh secara linear seiring dengan jumlah efek.
- Agen dunia wumpus dapat menghitung probabilitas untuk aspek dunia yang tidak teramati, sehingga menyempurnakan keputusan agen yang murni logis. Independensi bersyarat membuat perhitungan ini mudah dilakukan.

BAB 5

PENALARAN PROBABILISTIK

Bab 4 memperkenalkan elemen dasar teori probabilitas dan mencatat pentingnya hubungan independensi dan independensi bersyarat dalam menyederhanakan representasi probabilistik dunia. Bab ini memperkenalkan cara sistematis untuk merepresentasikan hubungan tersebut secara eksplisit dalam bentuk **jaringan Bayesian**. Kami mendefinisikan sintaksis dan semantik jaringan ini dan menunjukkan bagaimana mereka dapat digunakan untuk menangkap pengetahuan yang tidak pasti dengan cara yang alami dan efisien. Kami kemudian menunjukkan bagaimana inferensi probabilistik, meskipun secara komputasi sulit dilakukan dalam kasus terburuk, dapat dilakukan secara efisien dalam banyak situasi praktis. Kami juga menjelaskan berbagai algoritme inferensi perkiraan yang sering kali berlaku ketika inferensi eksak tidak layak. Kami akan menjelajahi cara-cara di mana teori probabilitas dapat diterapkan pada dunia dengan objek dan relasi—yaitu, pada representasi *first-order*, bukan *proporsional*. Terakhir, kami mensurvei pendekatan-pendekatan alternatif untuk penalaran yang tidak pasti.

5.1 MEWAKILI PENGETAHUAN DALAM DOMAIN YANG TAK PASTI

Dalam Bab 4, kita melihat bahwa distribusi probabilitas gabungan penuh dapat menjawab pertanyaan apa pun tentang domain, tetapi dapat menjadi sangat besar seiring bertambahnya jumlah variabel. Lebih jauh lagi, menentukan probabilitas untuk kemungkinan dunia satu per satu tidaklah wajar dan membosankan.

Kita juga melihat bahwa hubungan independensi dan independensi bersyarat di antara variabel dapat sangat mengurangi jumlah probabilitas yang perlu ditentukan untuk menentukan distribusi gabungan penuh. Bagian ini memperkenalkan struktur data yang disebut **jaringan Bayesian**¹ untuk merepresentasikan ketergantungan di antara variabel. Jaringan Bayesian pada dasarnya dapat merepresentasikan distribusi probabilitas gabungan penuh apa pun dan dalam banyak kasus dapat melakukannya dengan sangat ringkas.

Jaringan Bayesian adalah grafik terarah yang setiap simpulnya diberi anotasi dengan informasi probabilitas kuantitatif. Spesifikasi lengkapnya adalah sebagai berikut:

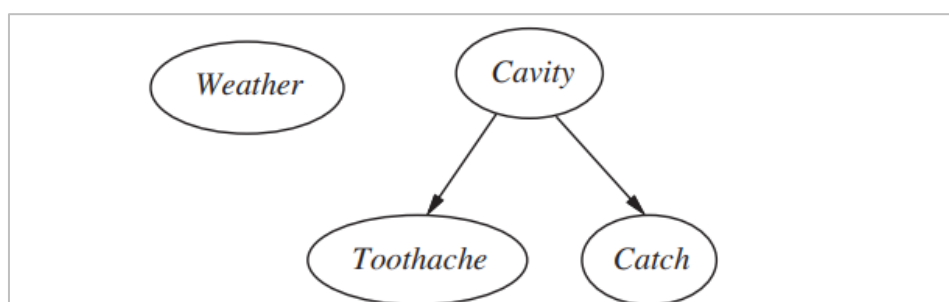
1. Setiap simpul sesuai dengan variabel acak, yang dapat berupa diskrit atau kontinu.
2. Seperangkat tautan atau anak panah terarah menghubungkan pasangan simpul. Jika ada anak panah dari simpul X ke simpul Y , X dikatakan sebagai *induk* dari Y . Grafik tersebut tidak memiliki siklus terarah (dan karenanya merupakan grafik asiklik terarah, atau DAG).
3. Setiap simpul X_i memiliki distribusi probabilitas bersyarat $P(X_i | \text{Parents}(X_i))$ yang mengukur pengaruh induk pada simpul tersebut.

Topologi jaringan—himpunan simpul dan tautan—menentukan hubungan independensi bersyarat yang berlaku dalam domain, dengan cara yang akan segera dijelaskan secara tepat.

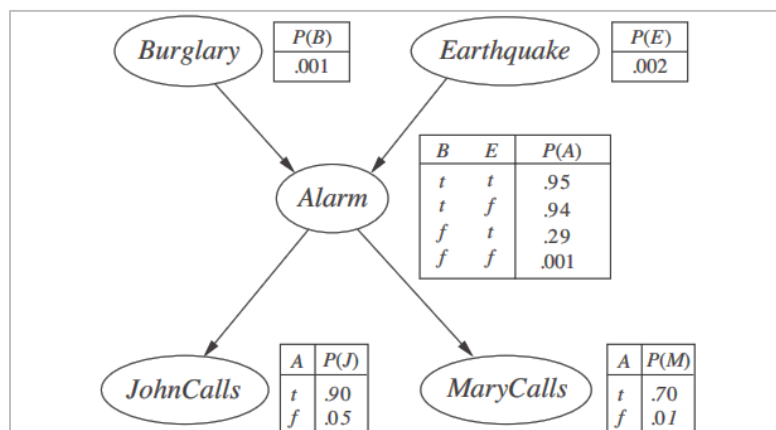
Arti *intuitif* dari anak panah biasanya adalah bahwa X memiliki *pengaruh langsung pada* Y , yang menunjukkan bahwa penyebab harus menjadi induk dari akibat. Biasanya mudah bagi pakar domain untuk memutuskan pengaruh langsung apa yang ada dalam domain—jauh lebih mudah, pada kenyataannya, daripada benar-benar menentukan probabilitas itu sendiri. Setelah topologi jaringan Bayesian ditata, kita hanya perlu menentukan distribusi probabilitas bersyarat untuk setiap variabel, yang diberikan induknya. Kita akan melihat bahwa kombinasi topologi dan distribusi bersyarat cukup untuk menentukan (secara implisit) distribusi gabungan penuh untuk semua variabel.

Ingat dunia sederhana yang dijelaskan dalam Bab 3, yang terdiri dari variabel *Toothache*, *Cavity*, *Catch*, dan *Weather*. Kami berpendapat bahwa *Weather* tidak bergantung pada variabel lain; lebih jauh, kami berpendapat bahwa *Toothache* dan *Catch* bersifat independen secara kondisional, mengingat *Cavity*. Hubungan ini direpresentasikan oleh struktur jaringan Bayesian yang ditunjukkan pada Gambar 5.1. Secara formal, independensi kondisional *Toothache* dan *Catch*, mengingat *Cavity*, ditunjukkan oleh tidak adanya hubungan antara *Toothache* dan *Catch*. Secara intuitif, jaringan tersebut merepresentasikan fakta bahwa *Cavity* merupakan penyebab langsung *Toothache* dan *Catch*, sedangkan tidak ada hubungan kausal langsung antara *Toothache* dan *Catch*.

Sekarang perhatikan contoh berikut, yang sedikit lebih rumit. Anda memiliki alarm antipencurian baru yang dipasang di rumah. Alarm tersebut cukup andal dalam mendeteksi pencurian, tetapi juga merespons Earthquake kecil sesekali. (Contoh ini berasal dari Judea Pearl, penduduk Los Angeles—maka dari itu minat yang mendalam pada Earthquake.) Anda juga memiliki dua tetangga, John dan Mary, yang telah berjanji untuk menelepon Anda di tempat kerja ketika mereka mendengar alarm. John hampir selalu menelepon ketika ia mendengar alarm, tetapi terkadang ia mengira telepon berdering dengan alarm dan menelepon saat itu juga. Mary, di sisi lain, menyukai musik yang agak keras dan sering kali tidak mendengar alarm sama sekali. Mengingat bukti siapa yang menelepon atau tidak, kami ingin memperkirakan kemungkinan terjadinya pencurian.



Gambar 5.1 Jaringan Bayesian sederhana di mana *Cuaca* tidak bergantung pada tiga variabel lain dan *Toothache* dan *Catch* bersifat independen bersyarat, dengan mempertimbangkan *Rongga*.



Gambar 5.2 Jaringan Bayesian yang umum, yang menunjukkan topologi dan tabel probabilitas bersyarat (CPT). Dalam CPT, huruf B, E, A, J , dan M masing-masing merupakan singkatan dari *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, dan *MaryCalls*.

Jaringan Bayesian untuk domain ini muncul pada Gambar 5.2. Struktur jaringan menunjukkan bahwa pencurian dan Earthquake secara langsung memengaruhi probabilitas alarm berbunyi, tetapi apakah John dan Mary akan menelepon hanya bergantung pada alarm. Dengan demikian, jaringan tersebut mewakili asumsi kami bahwa mereka tidak merasakan pencurian secara langsung, mereka tidak menyadari Earthquake kecil, dan mereka tidak berunding sebelum menelepon.

Distribusi kondisional pada Gambar 5.2 ditampilkan sebagai **tabel probabilitas kondisional**, atau CPT. (Bentuk tabel ini dapat digunakan untuk variabel diskrit; representasi lain, termasuk yang sesuai untuk variabel kontinu, dijelaskan dalam Bagian selanjutnya.) Setiap baris dalam CPT berisi probabilitas kondisional dari setiap nilai simpul untuk **kasus pengkondisian**. Kasus pengkondisian hanyalah kemungkinan kombinasi nilai untuk simpul induk—dunia miniatur yang mungkin, jika Anda suka. Setiap baris harus berjumlah 1, karena entri mewakili serangkaian kasus yang lengkap untuk variabel tersebut. Untuk variabel Boolean, setelah Anda mengetahui bahwa probabilitas nilai benar adalah p , probabilitas salah harus $1 - p$, jadi kita sering menghilangkan angka kedua, seperti pada Gambar 5.2. Secara umum, tabel untuk variabel Boolean dengan k induk Boolean berisi 2^k probabilitas yang dapat ditentukan secara independen. Node tanpa induk hanya memiliki satu baris, yang mewakili probabilitas sebelumnya dari setiap nilai variabel yang mungkin.

Perhatikan bahwa jaringan tidak memiliki node yang sesuai dengan Mary yang sedang mendengarkan musik keras atau telepon yang berdering dan membingungkan John. Faktor-faktor ini dirangkum dalam ketidakpastian yang terkait dengan tautan dari *Alarm* ke *JohnCalls* dan *MaryCalls*. Ini menunjukkan kemalasan dan ketidaktahuan dalam pengoperasian: akan banyak pekerjaan untuk mencari tahu mengapa faktor-faktor tersebut lebih atau kurang mungkin terjadi dalam kasus tertentu, dan kita tidak memiliki cara yang masuk akal untuk mendapatkan informasi yang relevan. Probabilitas sebenarnya merangkum serangkaian keadaan yang *berpotensi tak terbatas* di mana alarm mungkin gagal berbunyi (kelembapan tinggi, listrik mati, baterai mati, kabel putus, tikus mati tersangkut di dalam bel, dll.) atau John

atau Mary mungkin gagal menelepon dan melaporkannya (makan siang, berlibur, tuli sementara, helikopter lewat, dll.). Dengan cara ini, agen kecil dapat mengatasi dunia yang sangat besar, setidaknya secara perkiraan. Tingkat perkiraan dapat ditingkatkan jika kita memperkenalkan informasi relevan tambahan.

5.2 SEMANTIK JARINGAN BAYESIAN

Bagian sebelumnya menjelaskan apa itu jaringan, tetapi tidak menjelaskan apa artinya. Ada dua cara untuk memahami semantik jaringan Bayesian. Yang pertama adalah melihat jaringan sebagai representasi dari distribusi probabilitas gabungan. Yang kedua adalah melihatnya sebagai penyandian dari kumpulan pernyataan independensi bersyarat. Kedua pandangan tersebut setara, tetapi yang pertama ternyata membantu dalam memahami cara *membangun* jaringan, sedangkan yang kedua membantu dalam merancang prosedur inferensi.

Mewakili distribusi gabungan penuh

Jika dilihat sebagai bagian dari "sintaks", jaringan Bayesian adalah grafik asiklik berarah dengan beberapa parameter numerik yang dilampirkan ke setiap simpul. Salah satu cara untuk mendefinisikan apa arti jaringan—semantiknya—adalah dengan mendefinisikan cara jaringan tersebut merepresentasikan distribusi gabungan tertentu atas semua variabel. Untuk melakukan ini, pertama-tama kita perlu menarik kembali (sementara) apa yang kita katakan sebelumnya tentang parameter yang terkait dengan setiap simpul. Kami katakan bahwa parameter tersebut sesuai dengan probabilitas bersyarat $P(X_i | \text{Parents}(X_i))$; ini adalah pernyataan yang benar, tetapi sampai kita menetapkan semantik ke jaringan secara keseluruhan, kita harus menganggapnya hanya sebagai angka $\theta(X_i | \text{Parents}(X_i))$.

Entri generik dalam distribusi gabungan adalah probabilitas konjungsi penugasan tertentu ke setiap variabel, seperti $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$. Kami menggunakan notasi $P(x_1, \dots, x_n)$ sebagai singkatan untuk ini. Nilai entri ini diberikan oleh rumus

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta(x_i | \text{parents}(x_i)), \quad (14.1)$$

di mana $\text{parents}(X_i)$ menunjukkan nilai $\text{Parents}(X_i)$ yang muncul di $x_1, \dots, x_n$. Dengan demikian, setiap entri dalam distribusi gabungan direpresentasikan oleh produk elemen-elemen tabel probabilitas bersyarat (CPT) yang sesuai dalam jaringan Bayesian.

Dari definisi ini, mudah untuk membuktikan bahwa parameter $\theta(X_i | \text{Parents}(X_i))$ adalah probabilitas bersyarat $P(X_i | \text{Parents}(X_i))$ yang tersirat oleh distribusi gabungan (lihat Latihan 14.2). Oleh karena itu, kita dapat menulis ulang Persamaan (5.1) sebagai

$$P(x_1, \dots, x_n) = \prod_{i=1}^n \theta(x_i | \text{parents}(x_i)), \quad (14.2)$$

Dengan kata lain, tabel yang kita sebut tabel probabilitas bersyarat benar-benar merupakan tabel probabilitas bersyarat menurut semantik yang didefinisikan dalam Persamaan (5.1).

Untuk mengilustrasikan hal ini, kita dapat menghitung probabilitas bahwa alarm telah berbunyi, tetapi tidak terjadi pencurian atau Earthquake, dan John dan Mary sama-sama menelepon. Kita kalikan entri dari distribusi gabungan (menggunakan nama satu huruf untuk variabel):

$$\begin{aligned} P(j, m, a, \neg b, \neg e) &= P(j | a)P(m | a)P(a | \neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.000628. \end{aligned}$$

Bagian 13.3 menjelaskan bahwa distribusi gabungan penuh dapat digunakan untuk menjawab pertanyaan apa pun tentang domain. Jika jaringan Bayesian merupakan representasi dari distribusi gabungan, maka jaringan tersebut juga dapat digunakan untuk menjawab pertanyaan apa pun, dengan menjumlahkan semua entri gabungan yang relevan. Bagian 14.4 menjelaskan cara melakukannya, tetapi juga menjelaskan metode yang jauh lebih efisien.

Metode untuk membangun jaringan Bayesian

Persamaan (5.2) mendefinisikan apa arti jaringan Bayesian tertentu. Langkah berikutnya adalah menjelaskan cara *membangun* jaringan Bayesian sedemikian rupa sehingga distribusi gabungan yang dihasilkan merupakan representasi yang baik dari domain tertentu. Sekarang kami akan menunjukkan bahwa Persamaan (5.2) menyiratkan hubungan independensi bersyarat tertentu yang dapat digunakan untuk memandu perekrasan pengetahuan dalam membangun topologi jaringan. Pertama, kami menulis ulang entri dalam distribusi gabungan dalam bentuk probabilitas bersyarat, menggunakan aturan produk:

$$P(x_1, \dots, x_n) = P(x_n | x_{n-1}, \dots, x_1)P(x_{n-1}, \dots, x_1).$$

Kemudian kita ulangi prosesnya, dengan mengurangi setiap probabilitas konjungtif menjadi probabilitas bersyarat dan konjungtif yang lebih kecil. Kita akan mendapatkan satu produk besar:

$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_n | x_{n-1}, \dots, x_1)P(x_{n-1} | x_{n-2}, \dots, x_1) \cdots P(x_2 | x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i | x_{i-1}, \dots, x_1) \end{aligned}$$

Identitas ini disebut **aturan rantai**. Aturan ini berlaku untuk setiap himpunan variabel acak. Jika dibandingkan dengan Persamaan (5.2), kita melihat bahwa spesifikasi distribusi gabungan setara dengan pernyataan umum bahwa, untuk setiap variabel X_i dalam jaringan,

$$\mathbf{P}(X_i | X_{i-1}, \dots, x_1) = \mathbf{P}(X_i | \text{parents}(X_i)), \quad (14.3)$$

asalkan $Parents(X_i) \subseteq X_{i-1}, \dots, X_1$. Kondisi terakhir ini dipenuhi dengan penomoran simpul dengan cara yang konsisten dengan urutan parsial yang tersirat dalam struktur grafik.

Apa yang dikatakan Persamaan (5.3) adalah bahwa jaringan Bayesian adalah representasi domain yang benar hanya jika setiap simpul secara kondisional independen dari pendahulunya yang lain dalam urutan simpul, mengingat induknya. Kita dapat memenuhi kondisi ini dengan metodologi ini:

1. *Simpul*: Pertama-tama tentukan himpunan variabel yang diperlukan untuk memodelkan domain. Sekarang urutkan, $\{X_1, \dots, X_n\}$. Urutan apa pun akan berfungsi, tetapi jaringan yang dihasilkan akan lebih padat jika variabel diurutkan sedemikian rupa sehingga menyebabkan efek mendahului.
2. *Tautan*: Untuk $i = 1$ hingga n lakukan:
 - Pilih, dari $X_1, \dots, X_{i-1}$, himpunan minimal induk untuk X_i , sedemikian rupa sehingga Persamaan (5.3) terpenuhi.
 - Untuk setiap orangtua, masukkan tautan dari orangtua ke X_i .
 - CPT: Tuliskan tabel probabilitas bersyarat, $P(X_i | Parents(X_i))$.

Secara intuitif, induk dari simpul X_i harus berisi semua simpul tersebut di $X_1, \dots, X_{i-1}$ yang *secara langsung* memengaruhi X_i . Misalnya, anggaplah kita telah melengkapi jaringan pada Gambar 5.2 kecuali untuk pilihan induk untuk *MaryCalls*. *MaryCalls* tentu saja dipengaruhi oleh apakah ada *Burglary* atau *Earthquake*, tetapi tidak *dipengaruhi* secara langsung. Secara intuitif, pengetahuan kita tentang domain memberi tahu kita bahwa peristiwa-peristiwa ini memengaruhi perilaku panggilan Mary hanya melalui efeknya pada alarm. Selain itu, mengingat status alarm, apakah John menelepon tidak memiliki pengaruh pada panggilan Mary. Secara formal, kami percaya bahwa pernyataan independensi bersyarat berikut berlaku:

$$P(MaryCalls | JohnCalls, Alarm, Earthquake, Burglary) = P(MaryCalls | Alarm).$$

Dengan demikian, *Alarm* akan menjadi satu-satunya simpul induk untuk *MaryCalls*.

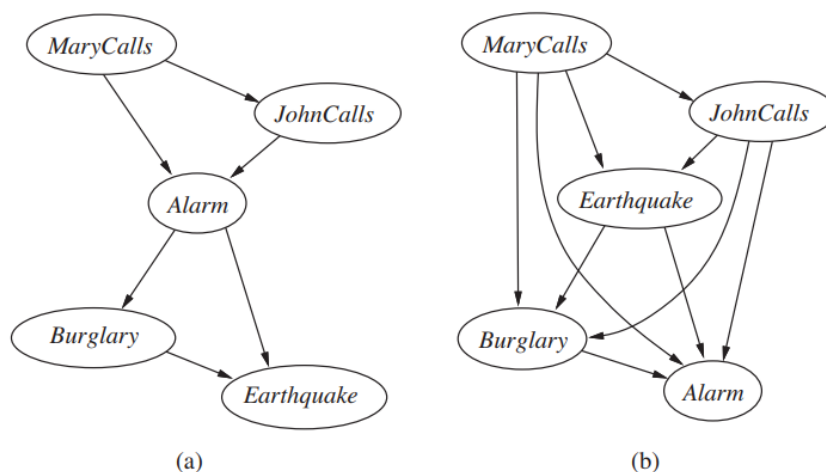
Karena setiap simpul hanya terhubung ke simpul sebelumnya, metode konstruksi ini menjamin bahwa jaringan tersebut asiklik. Properti penting lain dari jaringan Bayesian adalah bahwa jaringan tersebut tidak mengandung nilai probabilitas yang redundan. Jika tidak ada redundansi, maka *tidak ada kemungkinan untuk ketidakkonsistenan: tidak mungkin bagi insinyur pengetahuan atau pakar domain untuk membuat jaringan Bayesian yang melanggar aksioma probabilitas*.

Kekompakan dan urutan simpul

Selain menjadi representasi domain yang lengkap dan nonredundan, jaringan Bayesian sering kali jauh lebih *kompak* daripada distribusi gabungan penuh. Properti inilah yang memungkinkan untuk menangani domain dengan banyak variabel. Kekompakan jaringan Bayesian adalah contoh properti umum sistem **berstruktur lokal** (juga disebut **sparse**). SPARSE Dalam sistem berstruktur lokal, setiap subkomponen berinteraksi secara langsung hanya dengan sejumlah komponen lain yang dibatasi, terlepas dari jumlah total komponen. Struktur

lokal biasanya dikaitkan dengan pertumbuhan kompleksitas linear daripada eksponensial. Dalam kasus jaringan Bayesian, masuk akal untuk menganggap bahwa di sebagian besar domain, setiap variabel acak secara langsung dipengaruhi oleh paling banyak k variabel lain, untuk beberapa konstanta k . Jika kita mengasumsikan n variabel Boolean untuk kesederhanaan, maka jumlah informasi yang diperlukan untuk menentukan setiap tabel probabilitas bersyarat akan menjadi paling banyak 2^k angka, dan jaringan lengkap dapat ditentukan oleh $n2^k$ angka. Sebaliknya, distribusi gabungan berisi 2^n angka. Untuk membuatnya konkret, anggaplah kita memiliki $n = 30$ simpul, masing-masing dengan lima orang tua ($k = 5$). Maka jaringan Bayesian membutuhkan 960 angka, tetapi distribusi gabungan penuh membutuhkan lebih dari satu miliar.

Ada domain di mana setiap variabel dapat dipengaruhi secara langsung oleh semua variabel lainnya, sehingga jaringan terhubung sepenuhnya. Kemudian menentukan tabel probabilitas bersyarat memerlukan jumlah informasi yang sama seperti menentukan distribusi gabungan. Di beberapa domain, akan ada sedikit ketergantungan yang harus disertakan secara ketat dengan menambahkan tautan baru. Namun, jika ketergantungan ini lemah, maka kompleksitas tambahan dalam jaringan mungkin tidak sepadan dengan sedikit peningkatan akurasi. Misalnya, seseorang mungkin keberatan dengan jaringan pencurian kami dengan alasan bahwa jika terjadi *Earthquake*, maka John dan Mary tidak akan menelepon meskipun mereka mendengar alarm, karena mereka berasumsi bahwa *Earthquake* adalah penyebabnya. Apakah akan menambahkan tautan dari *Earthquake* ke *JohnCalls* dan *MaryCalls* (dan dengan demikian memperbesar tabel) bergantung pada perbandingan antara pentingnya mendapatkan probabilitas yang lebih akurat dengan biaya untuk menentukan informasi tambahan.



Gambar 5.3 Struktur jaringan bergantung pada urutan pengenalan. Di setiap jaringan, kami telah memperkenalkan node dalam urutan dari atas ke bawah.

Bahkan dalam domain yang terstruktur secara lokal, kita akan mendapatkan jaringan Bayesian yang ringkas hanya jika kita memilih urutan simpul dengan baik. Apa yang terjadi jika kita memilih urutan yang salah? Pertimbangkan lagi contoh pencurian. Misalkan kita

memutuskan untuk menambahkan simpul dalam urutan *MaryCalls*, *JohnCalls*, *Alarm*, *Burglary*, *Earthquake*. Kita kemudian mendapatkan jaringan yang agak lebih rumit yang ditunjukkan pada Gambar 5.3(a). Prosesnya berjalan sebagai berikut:

- Menambahkan *MaryCalls*: Tidak ada induk.
- Menambahkan *JohnCalls*: Jika Mary menelepon, itu mungkin berarti alarm telah berbunyi, yang tentu saja akan membuat John lebih mungkin menelepon. Oleh karena itu, *JohnCalls* membutuhkan *MaryCalls* sebagai induk.
- Menambahkan *Alarm*: Jelas, jika keduanya menelepon, lebih mungkin alarm telah berbunyi daripada jika hanya satu atau tidak ada yang menelepon, jadi kita membutuhkan *MaryCalls* dan *JohnCalls* sebagai induk.
- Menambahkan *Burglary*: Jika kita mengetahui status alarm, maka panggilan dari John atau Mary mungkin memberi kita informasi tentang dering telepon kita atau musik Mary, tetapi tidak tentang Burglary:

$$P(\text{Burglary} \mid \text{Alarm}, \text{JohnCalls}, \text{MaryCalls}) = P(\text{Burglary} \mid \text{Alarm}).$$

Oleh karena itu, kita hanya perlu *Alarm* sebagai induk.

- Menambahkan *Earthquake*: Jika alarm menyala, kemungkinan besar telah terjadi Earthquake. (Alarm adalah semacam detektor Earthquake.) Namun, jika kita mengetahui bahwa telah terjadi Burglary, maka itu menjelaskan alarm, dan kemungkinan Earthquake hanya akan sedikit di atas normal. Oleh karena itu, kita perlu *Alarm* dan *Burglary* sebagai induk.

Jaringan yang dihasilkan memiliki dua tautan lebih banyak daripada jaringan asli pada Gambar 5.2 dan memerlukan tiga probabilitas lagi untuk ditentukan. Yang lebih buruk, beberapa tautan menggambarkan hubungan yang renggang yang memerlukan penilaian probabilitas yang sulit dan tidak wajar, seperti menilai probabilitas *Earthquake*, mengingat *Burglary* dan *Alarm*. Fenomena ini cukup umum dan terkait dengan perbedaan antara model **kausal** dan **diagnostik** yang diperkenalkan di Bagian 13.5.1 (lihat juga Latihan 8.13). Jika kita mencoba membangun model diagnostik dengan tautan dari gejala ke penyebab (seperti dari *MaryCalls* ke *Alarm* atau *Alarm* ke *Burglary*), kita akhirnya harus menentukan ketergantungan tambahan antara penyebab yang sebaliknya independen (dan sering kali antara gejala yang terjadi secara terpisah juga). *Jika kita berpegang pada model kausal, kita akhirnya harus menentukan lebih sedikit angka, dan angka-angka akan sering lebih mudah muncul.* Dalam domain kedokteran, misalnya, telah ditunjukkan oleh Tversky dan Kahneman (1982) bahwa dokter ahli lebih suka memberikan penilaian probabilitas untuk aturan kausal daripada untuk yang diagnostik.

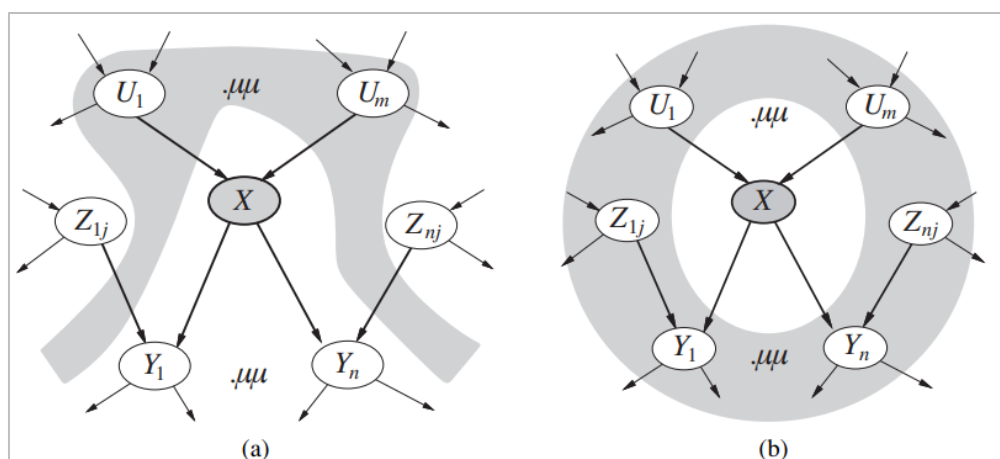
Gambar 5.3(b) menunjukkan urutan simpul yang sangat buruk: *MaryCalls*, *JohnCalls*, *Earthquake*, *Burglary*, *Alarm*. Jaringan ini memerlukan 31 probabilitas berbeda untuk ditetapkan—jumlah yang sama persis dengan distribusi gabungan penuh. Akan tetapi, penting untuk menyadari bahwa salah satu dari ketiga jaringan tersebut dapat merepresentasikan *distribusi gabungan yang sama persis*. Dua versi terakhir gagal merepresentasikan semua

hubungan independensi bersyarat dan karenanya malah menetapkan banyak angka yang tidak perlu.

Hubungan independensi bersyarat dalam jaringan Bayesian

Kami telah menyediakan semantik "numerik" untuk jaringan Bayesian dalam hal representasi distribusi gabungan penuh, seperti dalam Persamaan (5.2). Dengan menggunakan semantik ini untuk memperoleh metode untuk membangun jaringan Bayesian, kami dituntun pada konsekuensi bahwa sebuah simpul secara kondisional independen dari pendahulunya yang lain, mengingat induknya. Ternyata kita juga bisa pergi ke arah lain. Kita dapat mulai dari semantik "topologi" yang menentukan hubungan independensi kondisional yang dikodekan oleh struktur grafik, dan dari sini kita dapat memperoleh semantik "numerik". Semantik topologi² menentukan bahwa setiap variabel secara kondisional independen dari non-keturunannya, mengingat induknya. Misalnya, pada Gambar 5.2, *JohnCalls* independen dari *Burglary*, *Earthquake*, dan *MaryCalls* mengingat nilai *Alarm*. Definisi tersebut diilustrasikan pada Gambar 5.4(a). Dari pernyataan independensi bersyarat ini dan interpretasi parameter jaringan $\theta(X_i | \text{Parents}(X_i))$ sebagai spesifikasi probabilitas bersyarat $P(X_i | \text{Parents}(X_i))$, distribusi gabungan lengkap yang diberikan dalam Persamaan (5.2) dapat direkonstruksi. Dalam pengertian ini, semantik "numerik" dan semantik "topologis" adalah setara.

Sifat independensi penting lainnya tersirat oleh semantik topologis: sebuah simpul secara kondisional independen dari semua simpul lain dalam jaringan, jika diberikan induknya, anak-anaknya, dan orang tua dari anak-anaknya—yaitu, jika diberikan **selimut Markov**-nya. Misalnya, *Burglary* independen dari *JohnCalls* dan *MaryCalls*, jika diberikan *Alarm* dan *Earthquake*. Sifat ini diilustrasikan dalam Gambar 5.4(b).



Gambar 5.4 (a) Sebuah node X secara kondisional independen dari non-keturunannya (misalnya, Z_{ij} s) jika diberikan induknya (U_i s ditunjukkan di area abu-abu). (b) Sebuah node X secara kondisional independen dari semua node lain di jaringan jika diberikan selimut Markov-nya (area abu-abu).

5.3 REPRESENTASI DISTRIBUSI BERSYARAT YANG EFISIEN

Bahkan jika jumlah maksimum induk k cukup kecil, pengisian CPT untuk sebuah simpul memerlukan hingga $O(2^k)$ angka dan mungkin banyak pengalaman dengan semua kasus pengkondisian yang mungkin. Faktanya, ini adalah skenario terburuk di mana hubungan antara induk dan anak sepenuhnya bersifat arbitrer. Biasanya, hubungan tersebut dapat dijelaskan oleh **distribusi kanonik** yang sesuai dengan beberapa pola standar. Dalam kasus seperti itu, tabel lengkap dapat ditentukan dengan memberi nama pola dan mungkin menyediakan beberapa parameter—jauh lebih mudah daripada menyediakan sejumlah parameter eksponensial.

Contoh paling sederhana disediakan oleh **simpul deterministik**. Sebuah simpul deterministik memiliki nilai yang ditentukan secara tepat oleh nilai-nilai induknya, tanpa ketidakpastian. Hubungan tersebut dapat berupa hubungan yang logis: misalnya, hubungan antara simpul induk *Canadian*, *US*, *Mexican* dan simpul anak *NorthAmerican* hanyalah bahwa anak tersebut merupakan disjungsi dari kedua orangtuanya. Hubungan tersebut juga dapat berupa angka: misalnya, jika simpul induk adalah harga model mobil tertentu di beberapa dealer dan simpul anak adalah harga yang akhirnya dibayarkan oleh pemburu barang murah, maka simpul anak adalah nilai minimum dari nilai induk; atau jika simpul induk adalah aliran masuk danau (sungai, limpasan, presipitasi) dan aliran keluar (sungai, penguapan, rembesan) dan anak adalah perubahan permukaan air danau, maka nilai anak adalah jumlah dari induk aliran masuk dikurangi jumlah induk aliran keluar.

Hubungan yang tidak pasti sering kali dapat dicirikan oleh apa yang disebut hubungan logis yang **bising**. Contoh standar adalah hubungan **OR-bising**, yang merupakan generalisasi dari OR logis. Dalam logika proposisional, kita dapat mengatakan bahwa *Demam* benar jika dan hanya jika *Pilek*, *Flu*, atau *Malaria* benar. Model OR-berisik memungkinkan ketidakpastian tentang kemampuan masing-masing orangtua untuk menyebabkan anak menjadi benar—hubungan kausal antara orangtua dan anak mungkin *terhambat*, sehingga pasien bisa saja terkena flu, tetapi tidak menunjukkan demam. Model tersebut membuat dua asumsi. Pertama, model tersebut mengasumsikan bahwa semua kemungkinan penyebab tercantum. (Jika ada yang hilang, kita selalu dapat menambahkan apa yang disebut **simpul kebocoran** yang mencakup "penyebab lain-lain.") Kedua, model tersebut mengasumsikan bahwa penghambatan masing-masing orangtua tidak bergantung pada penghambatan orangtua lainnya: misalnya, apa pun yang menghambat *Malaria* dari menyebabkan demam tidak bergantung pada apa pun yang menghambat *Flu* dari menyebabkan demam. Dengan asumsi ini, *Demam* salah jika dan hanya jika semua orangtua *sejatinya* terhambat, dan probabilitas ini adalah hasil kali probabilitas penghambatan q untuk setiap orangtua. Mari kita asumsikan probabilitas penghambatan individual ini sebagai berikut:

$$\begin{aligned} q_{cold} &= P(\neg fever \mid cold, \neg flu, \neg malaria) = 0.6, \\ q_{flu} &= P(\neg fever \mid \neg cold, flu, \neg malaria) = 0.2, \\ q_{malaria} &= P(\neg fever \mid \neg cold, \neg flu, malaria) = 0.1. \end{aligned}$$

Kemudian, dari informasi ini dan asumsi-asumsi OR yang bising, seluruh CPT dapat dibangun. Aturan umumnya adalah bahwa

$$P(x_i | \text{parents}(X_i)) = 1 - \prod_{\{j: X_j = \text{true}\}} q_j$$

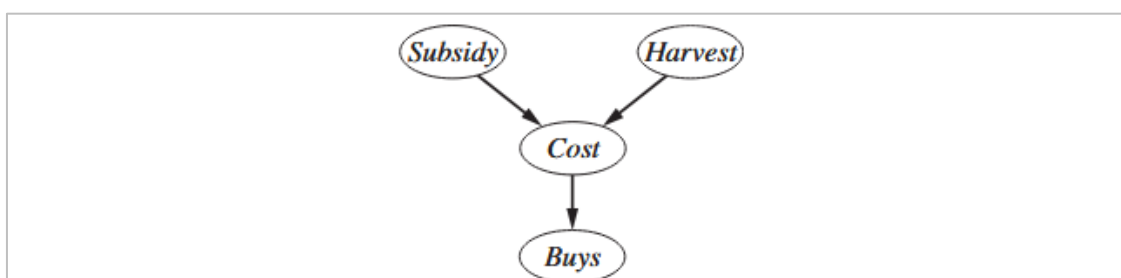
di mana produk diambil alih oleh induk yang ditetapkan menjadi benar untuk baris CPT tersebut. Tabel berikut mengilustrasikan perhitungan ini:

<i>Cold</i>	<i>Flu</i>	<i>Malaria</i>	<i>P(Fever)</i>	<i>P(¬Fever)</i>
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	$0.012 = 0.6 \times 0.2 \times 0.1$

Secara umum, hubungan logis yang bising di mana variabel bergantung pada k induk dapat dijelaskan menggunakan parameter $O(k)$ alih-alih $O(2^k)$ untuk tabel probabilitas bersyarat penuh. Hal ini membuat penilaian dan pembelajaran jauh lebih mudah. Misalnya, jaringan CPCS (Pradhan dan rekan rekan., 1994) menggunakan distribusi bising-OR dan bising-MAX untuk memodelkan hubungan di antara penyakit dan gejala dalam pengobatan internal. Dengan 448 simpul dan 906 tautan, hanya diperlukan 8.254 nilai, bukan 133.931.430 untuk jaringan dengan CPT penuh.

Jaringan Bayesian dengan variabel kontinu

Banyak masalah dunia nyata melibatkan kuantitas kontinu, seperti tinggi, massa, suhu, dan uang; pada kenyataannya, banyak statistik berurusan dengan variabel acak yang domainnya kontinu. Menurut definisi, variabel kontinu memiliki jumlah nilai yang mungkin tak terbatas, sehingga mustahil untuk menentukan probabilitas bersyarat secara eksplisit untuk setiap nilai. Salah satu cara yang mungkin untuk menangani variabel kontinu adalah menghindarinya dengan menggunakan **diskritisasi**—yaitu, membagi nilai-nilai yang mungkin ke dalam satu set interval yang tetap. Misalnya, suhu dapat dibagi menjadi ($<0^\circ\text{C}$), ($0^\circ\text{C} - 100^\circ\text{C}$), dan ($> 100^\circ\text{C}$).



Gambar 5.5 Jaringan sederhana dengan variabel diskrit (*Subsidi* dan *Pembelian*) dan variabel kontinyu (*Harvest* dan *Biaya*).

Diskritisasi terkadang merupakan solusi yang memadai, tetapi sering kali mengakibatkan hilangnya akurasi yang cukup besar dan CPT yang sangat besar. Solusi yang paling umum adalah mendefinisikan keluarga standar fungsi kepadatan probabilitas yang ditentukan oleh sejumlah **parameter** yang terbatas. Misalnya, distribusi Gaussian (atau normal) $N(\mu, \sigma^2)(x)$ memiliki rata-rata μ dan varians σ^2 sebagai parameter. Solusi lain—terkadang disebut representasi **nonparametrik**—adalah mendefinisikan distribusi kondisional secara implisit dengan kumpulan contoh, yang masing-masing berisi nilai-nilai spesifik dari variabel induk dan anak.

Jaringan dengan variabel diskrit dan kontinu disebut **jaringan Bayesian hibrida**. Untuk menentukan jaringan hibrida, kita harus menentukan dua jenis distribusi baru: distribusi kondisional untuk variabel kontinu yang diberikan induk diskrit atau kontinu; dan distribusi kondisional untuk variabel diskrit yang diberikan induk kontinu. Perhatikan contoh sederhana pada Gambar 5.5, di mana seorang pelanggan membeli beberapa buah tergantung pada biayanya, yang pada gilirannya tergantung pada ukuran Harvest dan apakah skema subsidi pemerintah beroperasi. Variabel *Biaya* bersifat kontinu dan memiliki induk kontinu dan diskrit; variabel *Membeli* bersifat diskrit dan memiliki induk kontinu.

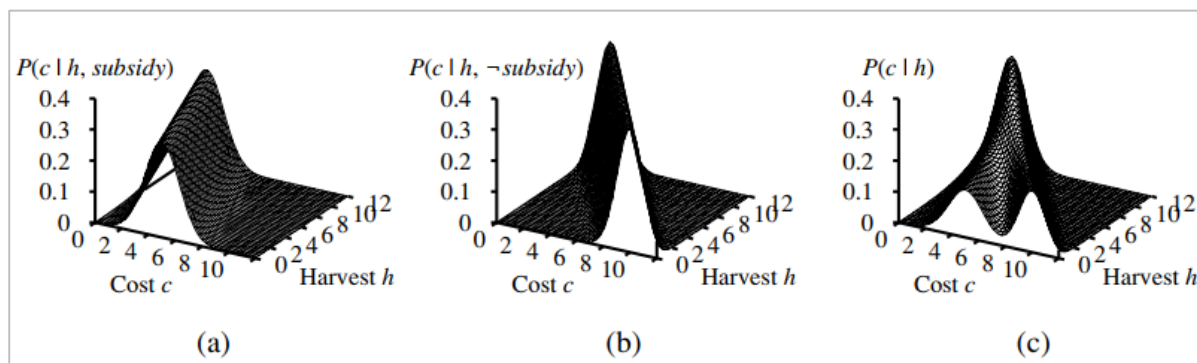
Untuk variabel *Biaya*, kita perlu menentukan $P(\text{Cost} \mid \text{Harvest}, \text{Subsidy})$. Induk diskrit ditangani dengan enumerasi—yaitu, dengan menentukan $P(\text{Cost} \mid \text{Harvest}, \text{Subsidy})$ dan $P(\text{Cost} \mid \text{Harvest}, \neg \text{Subsidy})$. Untuk menangani *Harvest*, kita tentukan bagaimana distribusi atas biaya c bergantung pada nilai kontinu h dari *Harvest*. Dengan kata lain, kita tentukan *parameter* distribusi biaya sebagai fungsi dari h . Pilihan yang paling umum adalah distribusi **Gaussian linear**, di mana anak memiliki distribusi Gaussian yang rata-rata μ bervariasi secara linear dengan nilai induk dan simpangan baku σ tetap. Kita memerlukan dua distribusi, satu untuk *subsidi* dan satu untuk $\neg \text{subsidi}$, dengan parameter yang berbeda:

$$P(c \mid h, \text{subsidi}) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{c - (a_t h + b_t)}{\sigma_t} \right)^2}$$

$$P(c \mid h, \neg \text{subsidi}) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{c - (a_f h + b_f)}{\sigma_f} \right)^2}.$$

Untuk contoh ini, distribusi kondisional untuk Biaya ditetapkan dengan memberi nama distribusi Gaussian linear dan menyediakan parameter di, $a_t, b_t, \sigma_t, a_f, b_f$, dan σ_f . Gambar 5.6(a) dan (b) menunjukkan kedua hubungan ini. Perhatikan bahwa dalam setiap kasus kemiringannya negatif, karena biaya menurun seiring dengan peningkatan pasokan. (Tentu saja, asumsi linearitas menyiratkan bahwa biaya menjadi negatif pada titik tertentu; model linear masuk akal hanya jika ukuran Harvest dibatasi pada rentang yang sempit.) Gambar 5.6(c) menunjukkan distribusi $P(c \mid h)$, yang dirata-ratakan pada dua kemungkinan nilai *Subsidi* dan

dengan asumsi bahwa masing-masing memiliki probabilitas awal 0,5. Ini menunjukkan bahwa bahkan dengan model yang sangat sederhana, distribusi yang cukup menarik dapat direpresentasikan.



Gambar 5.6 Grafik pada (a) dan (b) menunjukkan distribusi probabilitas atas *Biaya* sebagai fungsi dari ukuran *Harvest*, dengan *Subsidi* benar dan salah, berturut-turut. Grafik (c) menunjukkan distribusi $P(\text{Cost} | \text{Harvest})$, yang diperoleh dengan menjumlahkan kedua kasus subsidi.

Distribusi kondisional Gaussian linear memiliki beberapa sifat khusus. Jaringan yang hanya berisi variabel kontinu dengan distribusi Gaussian linear memiliki distribusi gabungan yang merupakan distribusi Gaussian multivariat atas semua variabel. Lebih jauh, distribusi posterior yang diberikan bukti apa pun juga memiliki sifat ini. Ketika variabel diskrit ditambahkan sebagai induk (bukan sebagai anak) dari variabel kontinu, jaringan mendefinisikan distribusi **Gaussian bersyarat**, atau CG: diberikan penugasan apa pun pada variabel diskrit, distribusi atas variabel kontinu adalah Gaussian multivariat.

Sekarang kita beralih ke distribusi untuk variabel diskrit dengan induk kontinu. Pertimbangkan, misalnya, simpul *Buys* pada Gambar 5.5. Tampaknya masuk akal untuk berasumsi bahwa pelanggan akan membeli jika biayanya rendah dan tidak akan membeli jika tinggi dan bahwa kemungkinan membeli bervariasi dengan lancar di beberapa wilayah perantara. Dengan kata lain, distribusi bersyarat seperti fungsi ambang batas "lunak". Salah satu cara untuk membuat ambang batas lunak adalah dengan menggunakan *integral* dari distribusi normal standar:

$$\Phi(x) = \int_{-\infty}^x N(0, 1)(x) dx .$$

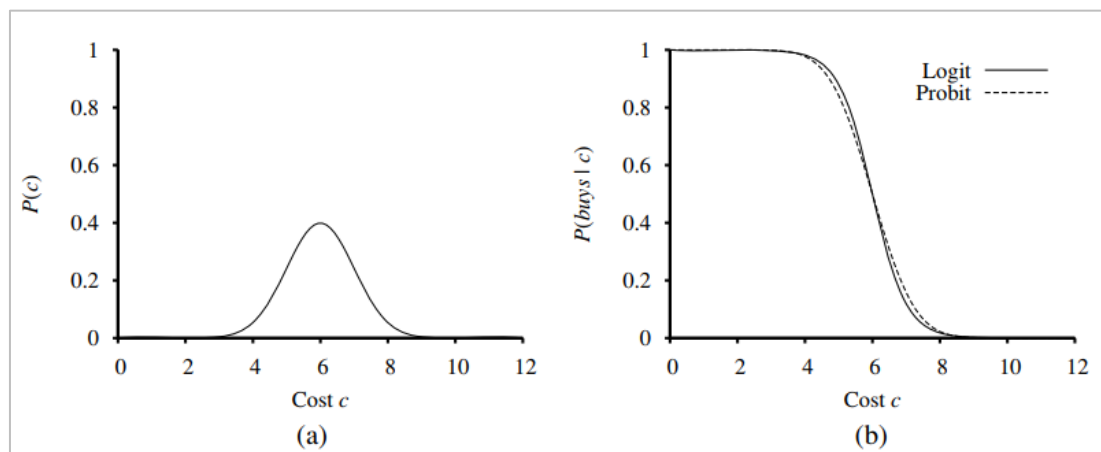
Maka kemungkinan *Pembelian* dengan *Biaya* yang diberikan mungkin adalah

$$P(\text{buys} | \text{Cost} = c) = \Phi((-c + \mu)/\sigma) ,$$

yang berarti bahwa ambang batas biaya terjadi di sekitar μ , lebar wilayah ambang batas sebanding dengan σ , dan probabilitas pembelian menurun seiring dengan peningkatan biaya.

Distribusi probit ini (diucapkan "pro-bit" dan kependekan dari "unit probabilitas")

diilustrasikan dalam Gambar 5.7(a). Bentuknya dapat dibenarkan dengan mengusulkan bahwa proses pengambilan keputusan yang mendasarinya memiliki ambang batas yang keras, tetapi lokasi ambang batas yang tepat tunduk pada gangguan Gaussian acak.



Gambar 5.7 (a) Distribusi normal (Gaussian) untuk ambang batas biaya, berpusat pada $\mu = 6,0$ dengan deviasi standar $\sigma = 1,0$. (b) Distribusi logit dan probit untuk probabilitas pembelian yang diberikan biaya, untuk parameter $\mu = 6,0$ dan $\sigma = 1,0$.

Alternatif untuk model probit adalah **distribusi logit** (diucapkan “low-jit”). Distribusi ini menggunakan **fungsi logistik** $1/(1 + e^{-x})$ untuk menghasilkan ambang batas lunak:

$$P(\text{buys} \mid \text{Cost} = c) = \frac{1}{1 + \exp\left(-2 \frac{-c + \mu}{\sigma}\right)}.$$

Hal ini diilustrasikan pada Gambar 5.7(b). Kedua distribusi tersebut tampak serupa, tetapi logit sebenarnya memiliki “ekor” yang jauh lebih panjang. Probit sering kali lebih sesuai dengan situasi nyata, tetapi logit terkadang lebih mudah ditangani secara matematis. Logit digunakan secara luas dalam jaringan saraf. Baik probit maupun logit dapat digeneralisasi untuk menangani beberapa induk kontinu dengan mengambil kombinasi linier dari nilai induk.

5.4 INFERENSI TEPAT DALAM JARINGAN BAYESIAN

Tugas dasar untuk setiap sistem inferensi probabilistik adalah menghitung distribusi probabilitas posterior untuk serangkaian *variabel kueri*, yang diberikan beberapa **kejadian** teramati—yaitu, beberapa penugasan nilai pada serangkaian **variabel bukti**. Untuk menyederhanakan penyajian, kami akan mempertimbangkan hanya satu variabel kueri pada satu waktu; algoritme dapat dengan mudah diperluas ke kueri dengan beberapa variabel. Kami akan menggunakan notasi dari Bab 13: X menunjukkan variabel kueri; E menunjukkan serangkaian variabel bukti $E_1, \dots, E_m$, dan $\mathbf{e}$ adalah kejadian teramati tertentu; $\mathbf{Y}$ akan menunjukkan variabel nonbukti, nonkueri $Y_1, \dots, Y_l$ (disebut **variabel tersembunyi**). Dengan

demikian, rangkaian variabel lengkap adalah $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. Kueri tipikal meminta distribusi probabilitas posterior $\mathbf{P}(X|\mathbf{e})$.

Dalam jaringan pencurian, kita mungkin mengamati kejadian di mana *JohnCalls* = *true* dan *MaryCalls* = *true*. Kita kemudian dapat menanyakan, misalnya, probabilitas bahwa pencurian telah terjadi:

$$\mathbf{P}(\text{Burglary} | \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = \langle 0.284, 0.716 \rangle.$$

Pada bagian ini, kami membahas algoritma eksak untuk menghitung probabilitas posterior dan akan mempertimbangkan kompleksitas tugas ini. Ternyata kasus umum tidak dapat dipecahkan, jadi Bagian selanjutnya membahas metode untuk inferensi perkiraan.

Inferensi dengan enumerasi

Bab 4 menjelaskan bahwa setiap probabilitas bersyarat dapat dihitung dengan menjumlahkan suku-suku dari distribusi gabungan penuh. Lebih khusus lagi, kueri $\mathbf{P}(X | \mathbf{e})$ dapat dijawab menggunakan Persamaan (4.9), yang kami ulangi di sini demi kenyamanan:

$$\mathbf{P}(X | \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_y \mathbf{P}(X, \mathbf{e}, y).$$

Sekarang, jaringan Bayesian memberikan representasi lengkap dari distribusi gabungan penuh. Lebih khusus lagi, Persamaan (5.2) menunjukkan bahwa suku $P(x, \mathbf{e}, y)$ dalam distribusi gabungan dapat ditulis sebagai hasil perkalian probabilitas bersyarat dari jaringan. Oleh karena itu, suatu kueri dapat dijawab menggunakan *jaringan Bayesian dengan menghitung jumlah hasil perkalian probabilitas bersyarat dari jaringan*.

Pertimbangkan kueri $\mathbf{P}(\text{Burglary} | \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true})$. Variabel tersembunyi untuk kueri ini adalah *Earthquake* dan *Alarm*. Dari Persamaan (4.9), menggunakan huruf awal untuk variabel guna mempersingkat ekspresi, kita memperoleh⁴

$$\mathbf{P}(B | j, m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_e \sum_a \mathbf{P}(B, j, m, e, a).$$

Semantik jaringan Bayesian (Persamaan (5.2)) kemudian memberi kita ekspresi dalam bentuk entri CPT. Untuk menyederhanakannya, kita lakukan ini hanya untuk *Burglary* = *benar*:

$$\mathbf{P}(B | j, m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a | b, e)P(j | a)P(m | a).$$

Untuk menghitung ekspresi ini, kita harus menambahkan empat suku, yang masing-masing dihitung dengan mengalikan lima angka. Dalam kasus terburuk, di mana kita harus menjumlahkan hampir semua variabel, kompleksitas algoritma untuk jaringan dengan n variabel Boolean adalah $O(n2^n)$.

Peningkatan dapat diperoleh dari pengamatan sederhana berikut: suku $P(b)$ adalah konstanta dan dapat dipindahkan ke luar penjumlahan atas a dan e , dan suku $P(e)$ dapat dipindahkan ke luar penjumlahan atas a . Oleh karena itu, kita memiliki

$$P(b | j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a | b, e) P(j | a) P(m | a). \quad (5.4)$$

Eksresi ini dapat dievaluasi dengan melakukan pengulangan melalui variabel secara berurutan, dengan mengalikan entri CPT saat kita melakukannya. Untuk setiap penjumlahan, kita juga perlu melakukan pengulangan atas nilai variabel yang mungkin. Struktur perhitungan ini ditunjukkan pada Gambar 5.8. Dengan menggunakan angka-angka dari Gambar 5.2, kita memperoleh $P(b | j, m) = \alpha \times 0,00059224$. Perhitungan yang sesuai untuk $\neg b$ menghasilkan $\alpha \times 0,0014919$; oleh karena itu,

$$P(B | j, m) = \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle.$$

Artinya, peluang terjadinya pencurian, jika ada panggilan dari kedua tetangga, adalah sekitar 28%.

Proses evaluasi untuk ekspresi dalam Persamaan (5.4) ditunjukkan sebagai pohon ekspresi dalam Gambar 5.8. Algoritma ENUMERATION-ASK dalam Gambar 5.9 mengevaluasi pohon tersebut menggunakan rekursi depth-first. Algoritma ini sangat mirip dalam strukturnya dengan algoritma backtracking untuk memecahkan CSP dan algoritma DPLL untuk kepuasan.

Kompleksitas ruang ENUMERATION-ASK hanya linear dalam jumlah variabel: algoritma tersebut menjumlahkan seluruh distribusi gabungan tanpa pernah membangunnya secara eksplisit. Sayangnya, kompleksitas waktunya untuk jaringan dengan n variabel Boolean selalu $O(2^n)$ —lebih baik daripada $O(n2^n)$ untuk pendekatan sederhana yang dijelaskan sebelumnya, tetapi masih agak suram.

Perhatikan bahwa pohon pada Gambar 5.8 membuat eksplisit *subekspresi berulang* yang dievaluasi oleh algoritma. Produk $P(j | a)P(m | a)$ dan $P(j | \neg a)P(m | \neg a)$ dihitung dua kali, sekali untuk setiap nilai e . Bagian berikutnya menjelaskan metode umum yang menghindari perhitungan yang sia-sia tersebut.

Algoritma eliminasi variabel

Algoritma enumerasi dapat ditingkatkan secara substansial dengan menghilangkan perhitungan berulang seperti yang diilustrasikan pada Gambar 5.8. Idennya sederhana: lakukan perhitungan sekali dan simpan hasilnya untuk digunakan nanti. Ini adalah bentuk pemrograman dinamis. Ada beberapa versi pendekatan ini; kami menyajikan algoritma **eliminasi variabel**, yang merupakan yang paling sederhana. Eliminasi variabel bekerja dengan mengevaluasi ekspresi seperti Persamaan (5.4) dalam urutan *right-to-left* (yaitu, dari *bottom up* pada Gambar 5.8). Hasil antara disimpan, dan penjumlahan atas setiap variabel dilakukan hanya untuk bagian-bagian ekspresi yang bergantung pada variabel tersebut.

Mari kita ilustrasikan proses ini untuk jaringan pencurian. Kita mengevaluasi ekspresi

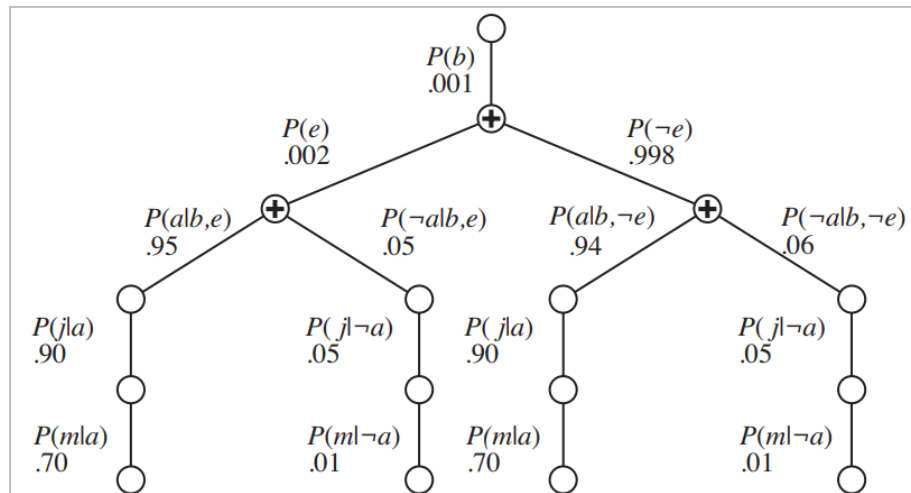
$$P(B | j, m) = \alpha \underbrace{P(B)}_{f_1(B)} \sum_e \underbrace{P(e)}_{f_2(E)} \sum_a \underbrace{P(a | B, e)}_{f_3(A, B, E)} \underbrace{P(j | a)}_{f_4(A)} \underbrace{P(m | a)}_{f_5(A)}.$$

Perhatikan bahwa kami telah memberi anotasi pada setiap bagian ekspresi dengan nama **faktor** yang sesuai; setiap faktor adalah matriks yang diindeks oleh nilai variabel argumennya. Misalnya, faktor $f_4(A)$ dan $f_5(A)$ yang sesuai dengan $P(j | a)$ dan $P(m | a)$ bergantung hanya pada A karena J dan M ditetapkan oleh kueri. Oleh karena itu, keduanya adalah vektor dua elemen:

$$f_4(A) = \begin{pmatrix} P(j | a) \\ P(j | \neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix} \quad f_5(A) = \begin{pmatrix} P(m | a) \\ P(m | \neg a) \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.01 \end{pmatrix}.$$

$f_3(A, B, E)$ akan menjadi matriks $2 \times 2 \times 2$, yang sulit ditampilkan pada halaman cetak. (Elemen “pertama” diberikan oleh $P(a | b, e) = 0,95$ dan “terakhir” oleh $P(\neg a | \neg b, \neg e) = 0,999$.) Dalam hal faktor, ekspresi kueri ditulis sebagai

$$P(B | j, m) = \alpha f_1(B) \times \sum_e f_2(E) \times \sum_a f_3(A, B, E) \times f_4(A) \times f_5(A)$$



Gambar 5.8 Struktur ekspresi yang ditunjukkan dalam Persamaan (5.4). Evaluasi dilakukan dari atas ke bawah, mengalikan nilai di sepanjang setiap jalur dan menjumlahkannya di simpul “+”. Perhatikan pengulangan jalur untuk j dan m .

```

function ENUMERATION – ASK( $X, e, bn$ ) returns a distribution over  $X$ 
  inputs:  $X$ , the query variable
            $e$ , observed values for variables  $E$ 
            $bn$ , a Bayes net with variables  $\{X\} \cup E \cup Y$  /*  $Y$  = hidden variables */

   $Q(X) \leftarrow$  a distribution over  $X$ , initially empty
  for each value  $x_i$  of  $X$  do
     $Q(x_i) \leftarrow$  ENUMERATE – ALL( $bn.VARS, e_{xi}$ )
    where  $e_{xi}$  is  $e$  extended with  $X = x_i$ 
  return NORMALIZE( $Q(X)$ )

function ENUMERATE – ALL( $vars, e$ ) returns a real number
  if EMPTY? ( $vars$ ) then return 1.0
   $Y \leftarrow$  FIRST( $vars$ )
  if  $Y$  has value  $y$  in  $e$ 
    then return  $P(y|parents(Y)) \times$  ENUMERATE – ALL( $REST(vars), e$ )
    else return  $\sum_y P(y|parents(Y)) \times$  ENUMERATE – ALL( $REST(vars), e_y$ )
    where  $e_y$  is  $e$  extended with  $Y = y$ 

```

Gambar 5.9 Algoritma enumerasi untuk menjawab pertanyaan pada jaringan Bayesian.

di mana operator “ $\times$ ” bukanlah perkalian matriks biasa, melainkan operasi perkalian **titik demi titik**, yang akan dijelaskan sebentar lagi.

Proses evaluasi adalah proses menjumlahkan variabel (dari kanan ke kiri) dari perkalian titik demi titik faktor untuk menghasilkan faktor baru, yang akhirnya menghasilkan faktor yang merupakan solusi, yaitu distribusi posterior atas variabel kueri. Langkah-langkahnya adalah sebagai berikut:

- Pertama, kita menjumlahkan A dari perkalian f_3 , f_4 , dan f_5 . Ini memberi kita faktor 2×2 baru $f_6(B, E)$ yang indeksnya berkisar hanya pada B dan E :

$$\begin{aligned}
 f_6(B, E) &= \sum_a f_3(A, B, E) \times f_4(A) \times f_5(A) \\
 &= (f_3(a, B, E) \times f_4(a) \times f_5(a)) + (f_3(\neg a, B, E) \times f_4(\neg a) \times f_5(\neg a))
 \end{aligned}$$

Sekarang kita tinggal dengan ekspresi

$$P(B | j, m) = \alpha f_1(B) \times \sum_e f_2(E) \times f_6(B, E).$$

- Selanjutnya, kita jumlahkan E dari hasil perkalian f_2 dan f_6 :

$$f_7(B) = \sum_e f_2(B, E)$$

$$= f_2(e) \times f_6(B, e) + f_2(\neg e) \times f_6(B, \neg e).$$

Hal ini meninggalkan ekspresi

$$P(B | j, m) = \alpha f_1(B) \times f_7(B)$$

yang dapat dievaluasi dengan mengambil produk titik demi titik dan menormalkan hasilnya.

Dengan memeriksa urutan ini, kita melihat bahwa dua operasi komputasi dasar diperlukan: produk titik demi titik dari sepasang faktor, dan menjumlahkan variabel dari produk faktor. Bagian berikutnya menjelaskan masing-masing operasi ini.

Operasi pada faktor

Produk titik demi titik dari dua faktor f_1 dan f_2 menghasilkan faktor f baru yang variabelnya merupakan *penggabungan* dari variabel dalam f_1 dan f_2 dan yang elemennya diberikan oleh produk dari elemen yang sesuai dalam dua faktor. Misalkan kedua faktor memiliki variabel $Y_1, \dots, Y_k$ yang sama. Maka kita memiliki

$$f(X_1 \dots X_j, Y_1 \dots Y_k, Z_1 \dots Z_l) = f_1(X_1 \dots X_j, Y_1 \dots Y_k) f_2(Y_1 \dots Y_k, Z_1 \dots Z_l)$$

Jika semua variabelnya biner, maka f_1 dan f_2 masing-masing memiliki 2^{j+k} dan 2^{k+l} entri, dan perkalian titik demi titik memiliki 2^{j+k+l} entri. Misalnya, jika diberikan dua faktor $f_1(A, B)$ dan $f_2(A, B)$, perkalian titik demi titik $f_1 \times f_2 = f_3(A, B, C)$ memiliki $2^{1+1+1} = 8$ entri, seperti yang diilustrasikan pada Gambar 5.10. Perhatikan bahwa faktor yang dihasilkan dari perkalian titik demi titik dapat berisi lebih banyak variabel daripada faktor mana pun yang dikalikan dan bahwa ukuran suatu faktor bersifat eksponensial dalam jumlah variabel. Di sinilah kompleksitas ruang dan waktu muncul dalam algoritma eliminasi variabel.

Tabel 14.10 Mengilustrasikan perkalian titik demi titik: $f_1(A, B) \times f_2(B, C) = f_3(A, B, C)$.									
A	B	$f_1(A, B)$	B	C	$f_2(B, C)$	A	B	C	$f_3(A, B, C)$
T	T	0.3	T	T	0.2	T	T	T	$0.3 \times 0.2 = 0.06$
T	F	0.7	T	F	0.8	T	T	F	$0.3 \times 0.8 = 0.24$
F	T	0.9	F	T	0.6	T	F	T	$0.7 \times 0.6 = 0.42$
F	F	0.1	F	F	0.4	T	F	F	$0.7 \times 0.4 = 0.28$
						F	T	T	$0.9 \times 0.2 = 0.18$
						F	T	F	$0.9 \times 0.8 = 0.72$

						F	F	T	$0.1 \times 0.6 =$
									0.06
						F	F	F	$0.1 \times 0.4 =$
									0.04

Penjumlahan suatu variabel dari hasil perkalian faktor dilakukan dengan menjumlahkan submatriks yang dibentuk dengan cara menetapkan variabel tersebut ke setiap nilainya secara bergantian. Misalnya, untuk menjumlahkan A dari $f_3(A, B, C)$, kita tulis

$$\begin{aligned} f(B, C) &= \sum_a f_3(A, B, C) = f_3(a, B, C) + f_3(\neg a, B, C) \\ &= \begin{pmatrix} .06 & .24 \\ .42 & .28 \end{pmatrix} + \begin{pmatrix} .18 & .72 \\ .06 & .04 \end{pmatrix} = \begin{pmatrix} .24 & .96 \\ .48 & .32 \end{pmatrix}. \end{aligned}$$

Satu-satunya triknya adalah dengan memperhatikan bahwa faktor apa pun yang *tidak* bergantung pada variabel yang akan dijumlahkan dapat dipindahkan ke luar penjumlahan. Misalnya, jika kita menjumlahkan E terlebih dahulu dalam jaringan pencurian, bagian ekspresi yang relevan akan menjadi

$$\sum_e f_2(E) \times f_3(A, B, E) \times f_4(A) \times f_5(A) = f_4(A) \times f_5(A) \times \sum_e f_2(E) \times f_3(A, B, E).$$

Sekarang produk perkalian titik demi titik di dalam penjumlahan dihitung, dan variabel dijumlahkan dari matriks yang dihasilkan.

Perhatikan bahwa matriks *tidak* dikalikan sampai kita perlu menjumlahkan variabel dari produk akumulasi. Pada titik itu, kita hanya mengalikan matriks yang menyertakan variabel yang akan dijumlahkan. Dengan fungsi yang diberikan untuk produk perkalian titik demi titik dan penjumlahan, algoritma eliminasi variabel itu sendiri dapat ditulis dengan cukup sederhana, seperti yang ditunjukkan pada Gambar 5.11.

Pengurutan variabel dan relevansi variabel

Algoritma pada Gambar 5.11 menyertakan fungsi ORDER yang tidak ditentukan untuk memilih urutan variabel. Setiap pilihan urutan menghasilkan algoritma yang valid, tetapi urutan yang berbeda menyebabkan faktor perantara yang berbeda dihasilkan selama perhitungan. Misalnya, dalam perhitungan yang ditunjukkan sebelumnya, kita menghilangkan A sebelum E ; jika kita melakukannya dengan cara lain, perhitungan menjadi

$$P(B|j, m) = \alpha f_1(B) \times \sum_a f_4(A) \times f_5(A) \times \sum_e f_2(E) \times f_3(A, B, E),$$

selama faktor baru $f_6(A, B)$ akan dihasilkan.

Secara umum, persyaratan waktu dan ruang untuk eliminasi variabel didominasi oleh ukuran faktor terbesar yang dibangun selama pengoperasian algoritma. Hal ini pada gilirannya ditentukan oleh urutan eliminasi variabel dan oleh struktur jaringan. Ternyata sulit untuk menentukan urutan optimal, tetapi beberapa heuristik yang baik tersedia. Salah satu metode yang cukup efektif adalah metode greedy: eliminasi variabel mana pun yang meminimalkan ukuran faktor berikutnya yang akan dibangun.

```
function ELIMINATION – ASK( $X, e, bn$ ) returns a distribution over  $X$ 
inputs:  $X$ , the query variable
          $e$ , observed values for variables  $E$ 
          $bn$ , a Bayesian network specifying joint distribution  $P(X_1, \dots, X_n)$ 

 $factors \leftarrow []$ 
for each  $var$  in ORDER( $bn.VARS$ ) do
     $factors \leftarrow [MAKE - FACTOR(var, e) | factors]$ 
    if  $var$  is a hidden variable then  $factors \leftarrow SUM - OUT(var, factors)$ 
return NORMALIZE(POINTWISE – PRODUCT( $factors$ ))
```

Gambar 5.11 Algoritma eliminasi variabel untuk inferensi dalam jaringan Bayesian.

Mari kita pertimbangkan satu kueri lagi: $P(JohnCalls \mid Burglary = true)$. Seperti biasa, langkah pertama adalah menuliskan penjumlahan bersarang:

$$P(J \mid b) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(J \mid a) \sum_m P(m \mid a).$$

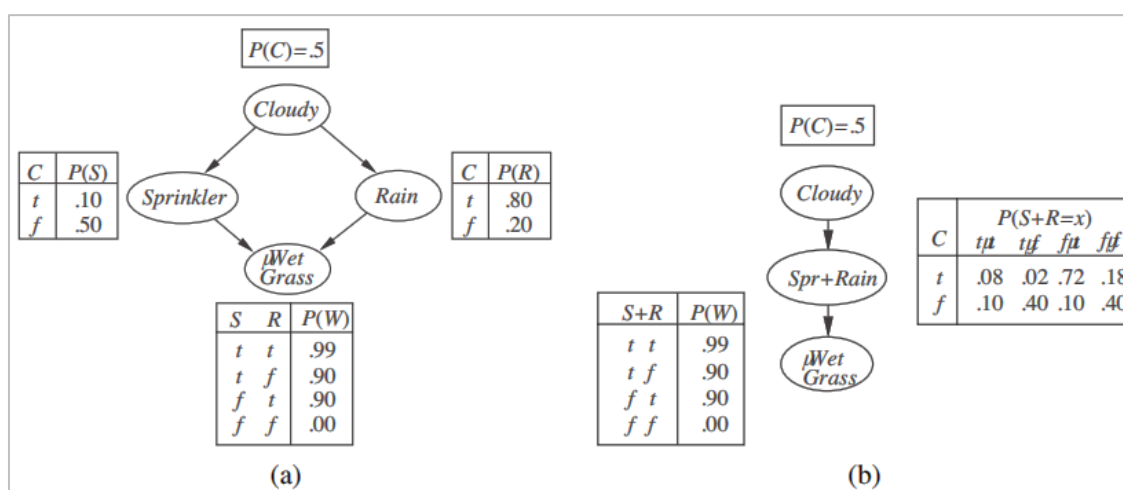
Jika kita mengevaluasi ekspresi ini dari kanan ke kiri, kita melihat sesuatu yang menarik: $\sum_m P(m \mid a)$ sama dengan 1 menurut definisi! Oleh karena itu, tidak perlu memasukkannya sejak awal; variabel M tidak relevan dengan kueri ini. Cara lain untuk mengatakan ini adalah bahwa hasil kueri $P(JohnCalls \mid Burglary = true)$ tidak berubah jika kita menghapus *MaryCalls* dari jaringan secara keseluruhan. Secara umum, kita dapat menghapus simpul daun apa pun yang bukan variabel kueri atau variabel bukti. Setelah penghapusannya, mungkin ada beberapa simpul daun lagi, dan ini juga mungkin tidak relevan. Melanjutkan proses ini, kita akhirnya menemukan bahwa *setiap variabel yang bukan leluhur variabel kueri atau variabel bukti tidak relevan dengan kueri*. Oleh karena itu, algoritme eliminasi variabel dapat menghapus semua variabel ini sebelum mengevaluasi kueri.

Kompleksitas inferensi eksak

Kompleksitas inferensi eksak dalam jaringan Bayesian sangat bergantung pada struktur jaringan. Jaringan pembobolan pada Gambar 5.2 termasuk dalam keluarga jaringan di mana paling banyak ada satu jalur tak berarah antara dua simpul mana pun dalam jaringan. Ini disebut jaringan **terhubung tunggal** atau **polytree**, dan mereka memiliki properti yang sangat bagus: *Kompleksitas waktu dan ruang dari inferensi eksak dalam polytree bersifat linear dalam*

ukuran jaringan. Di sini, ukuran didefinisikan sebagai jumlah entri CPT; jika jumlah induk dari setiap node dibatasi oleh konstanta, maka kompleksitasnya juga akan linear dalam jumlah node.

Untuk jaringan **terhubung berganda**, seperti pada Gambar 5.12(a), eliminasi variabel dapat memiliki kompleksitas waktu dan ruang eksponensial dalam kasus terburuk, bahkan ketika jumlah induk per node dibatasi. Ini tidak mengherankan ketika seseorang mempertimbangkan bahwa *karena ini mencakup inferensi dalam logika proposisional sebagai kasus khusus, inferensi dalam jaringan Bayesian bersifat NP-keras*. Faktanya, dapat ditunjukkan bahwa masalahnya sama sulitnya dengan menghitung jumlah penugasan yang memuaskan untuk rumus logika proposisional. Ini berarti bahwa hal tersebut #P-hard (“number-P hard”)—yaitu, lebih sulit daripada masalah NP-lengkap.



Gambar 5.12 (a) Jaringan terhubung jamak dengan tabel probabilitas bersyarat. (b) Ekuivalen berkelompok dari jaringan terhubung jamak.

Ada hubungan erat antara kompleksitas inferensi jaringan Bayesian dan kompleksitas masalah pemenuhan kendala (CSP). Kesulitan memecahkan CSP diskret terkait dengan seberapa “mirip pohon” grafik kendalanya. Ukuran seperti **lebar pohon**, yang membatasi kompleksitas pemecahan CSP, juga dapat diterapkan secara langsung ke jaringan Bayesian. Selain itu, algoritma eliminasi variabel dapat digeneralisasi untuk memecahkan CSP serta jaringan Bayesian.

Algoritma pengelompokan

Algoritma eliminasi variabel sederhana dan efisien untuk menjawab pertanyaan individual. Namun, jika kita ingin menghitung probabilitas posterior untuk semua variabel dalam jaringan, algoritma ini bisa jadi kurang efisien. Misalnya, dalam jaringan polytree, seseorang perlu mengeluarkan $O(n)$ pertanyaan yang masing-masing menghabiskan biaya $O(n)$, dengan total waktu $O(n^2)$. Dengan menggunakan algoritma **pengelompokan** (juga dikenal sebagai algoritma **join tree**), waktu dapat dikurangi menjadi $O(n)$. Karena alasan ini, algoritma ini banyak digunakan dalam perangkat jaringan Bayesian komersial.

Ide dasar pengelompokan adalah menggabungkan simpul-simpul individual jaringan untuk membentuk simpul-simpul kluster sedemikian rupa sehingga jaringan yang dihasilkan adalah polytree. Misalnya, jaringan yang terhubung secara berganda yang ditunjukkan pada Gambar 5.12(a) dapat diubah menjadi polytree dengan menggabungkan simpul *Sprinkler* dan *Rain* menjadi simpul kluster yang disebut *Sprinkler+Rain*, seperti yang ditunjukkan pada Gambar 5.12(b). Dua simpul Boolean digantikan oleh "meganode" yang memiliki empat kemungkinan nilai: *tt*, *tf*, *ft*, dan *ff*. Meganode hanya memiliki satu induk, variabel Boolean *Cloudy*, jadi ada dua kasus pengkondisian. Meskipun contoh ini tidak menunjukkannya, proses pengelompokan sering kali menghasilkan meganode yang memiliki beberapa variabel yang sama.

Setelah jaringan berbentuk polytree, diperlukan algoritma inferensi tujuan khusus, karena metode inferensi biasa tidak dapat menangani meganode yang berbagi variabel satu sama lain. Pada dasarnya, algoritma tersebut merupakan bentuk propagasi kendala (lihat Bab 6) di mana kendala memastikan bahwa meganode tetangga menyetujui probabilitas posterior dari setiap variabel yang sama. Dengan pencatatan yang cermat, algoritma ini mampu menghitung probabilitas posterior untuk semua node non-bukti dalam jaringan dalam waktu *linier* dalam ukuran jaringan terkluster. Namun, kesulitan NP dari masalah tersebut belum hilang: jika jaringan memerlukan waktu dan ruang eksponensial dengan eliminasi variabel, maka CPT dalam jaringan terkluster tentu akan menjadi besar secara eksponensial.

5.5 INFERENSI PERKIRAAN DALAM JARINGAN BAYESIAN

Mengingat inferensi eksak yang sulit dalam jaringan besar yang terhubung secara berganda, penting untuk mempertimbangkan metode inferensi perkiraan. Bagian ini menjelaskan algoritma pengambilan sampel acak, yang juga disebut algoritma **Monte Carlo**, yang memberikan jawaban perkiraan yang akurasinya bergantung pada jumlah sampel yang dihasilkan. Algoritma Monte Carlo, yang salah satu contohnya adalah simulasi anil, digunakan dalam banyak cabang ilmu pengetahuan untuk memperkirakan kuantitas yang sulit dihitung secara tepat. Dalam bagian ini, kami tertarik pada pengambilan sampel yang diterapkan pada perhitungan probabilitas posterior. Kami menjelaskan dua keluarga algoritma: pengambilan sampel langsung dan pengambilan sampel rantai Markov. Dua pendekatan lain—metode variasional dan perambatan loop—disebutkan dalam catatan di akhir bab.

Metode pengambilan sampel langsung

Elemen primitif dalam setiap algoritma pengambilan sampel adalah pembuatan sampel dari distribusi probabilitas yang diketahui. Misalnya, koin yang tidak bias dapat dianggap sebagai variabel acak *Koin* dengan nilai $\langle \text{kepala}, \text{ekor} \rangle$ dan distribusi prior $\mathbf{P}(\text{Koin}) = \langle 0,5, 0,5 \rangle$. Pengambilan sampel dari distribusi ini sama persis seperti melempar koin: dengan probabilitas 0,5 akan menghasilkan sisi *kepala*, dan dengan probabilitas 0,5 akan menghasilkan sisi *ekor*. Dengan sumber bilangan acak yang terdistribusi secara seragam dalam rentang $[0, 1]$, pengambilan sampel distribusi apa pun pada satu variabel, baik diskrit maupun kontinu, merupakan hal yang mudah.

Jenis proses pengambilan sampel acak yang paling sederhana untuk jaringan Bayesian menghasilkan kejadian dari jaringan yang tidak memiliki bukti yang terkait dengannya. Identy adalah mengambil sampel setiap variabel secara bergiliran, dalam urutan topologi. Distribusi probabilitas tempat nilai diambil sampelnya dikondisikan pada nilai yang telah ditetapkan ke induk variabel. Algoritma ini ditunjukkan pada Gambar 5.13. Kita dapat mengilustrasikan operasinya pada jaringan pada Gambar 5.12(a), dengan asumsi urutan [*Berawan, Penyiram, Hujan, Rumput Basah*]:

1. Pengambilan sampel dari $\mathbf{P}(\textit{Cloudy}) = \langle 0,5, 0,5 \rangle$, nilainya *benar*.
2. Sampel dari $\mathbf{P}(\textit{Sprinkler} \mid \textit{Cloudy} = \textit{true}) = \langle 0,1, 0,9 \rangle$, nilai *salah*.
3. Sampel dari $\mathbf{P}(\textit{Rain} \mid \textit{Cloudy} = \textit{true}) = \langle 0,8, 0,2 \rangle$, nilai *benar*.
4. Sampel dari $\mathbf{P}(\textit{WetGrass} \mid \textit{Sprinkler} = \textit{false}, \textit{Rain} = \textit{true}) = \langle 0,9, 0,1 \rangle$, nilai *benar*.

Dalam kasus ini, PRIOR-SAMPLE mengembalikan kejadian [*true, false, true, true*].

function PRIOR – SAMPLE(*bn*) **returns** an event sampled from the prior specified by *bn*
inputs: *bn*, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \dots, X_n)$

x $\leftarrow$ an event with *n* elements
foreach variable X_i in $X_1, \dots, X_n$ **do**
x[*i*] $\leftarrow$ a random sample from $\mathbf{P}(X_i \mid \textit{parents}(X_i))$
return *x*

Gambar 5.13 Algoritma pengambilan sampel yang menghasilkan peristiwa dari jaringan Bayesian. Setiap variabel diambil sampelnya menurut distribusi kondisional yang diberikan nilai yang telah diambil sampelnya untuk induk variabel tersebut.

Mudah untuk melihat bahwa PRIOR-SAMPLE menghasilkan sampel dari distribusi gabungan sebelumnya yang ditentukan oleh jaringan. Pertama, misalkan $S_{PS}(x_1, \dots, x_n)$ adalah probabilitas bahwa suatu peristiwa tertentu dihasilkan oleh algoritma PRIOR-SAMPLE. Hanya dengan melihat proses pengambilan sampel, kita memiliki

$$S_{PS}(x_1 \dots x_n) = \prod_{i=1}^n P(x_i \mid \textit{parents}(X_i))$$

karena setiap langkah pengambilan sampel hanya bergantung pada nilai induk. Ekspresi ini seharusnya terlihat familiar, karena ini juga merupakan probabilitas kejadian menurut representasi jaringan Bayesian dari distribusi gabungan, seperti yang dinyatakan dalam Persamaan (5.2). Artinya, kita memiliki

$$S_{PS}(x_1 \dots x_n) = P(x_1 \dots x_n)$$

Fakta sederhana ini memudahkan untuk menjawab pertanyaan dengan menggunakan sampel.

Dalam algoritma pengambilan sampel apa pun, jawaban dihitung dengan menghitung sampel aktual yang dihasilkan. Misalkan ada N sampel total, dan misalkan $NPS(x_1, \dots, x_n)$ adalah jumlah kali kejadian spesifik $x_1, \dots, x_n$ terjadi dalam kumpulan sampel. Kami mengharapkan angka ini, sebagai pecahan dari total, untuk bertemu dalam batas ke nilai yang diharapkan menurut probabilitas pengambilan sampel:

$$\lim_{n \rightarrow \infty} \frac{NPS(x_1 \dots x_n)}{N} = S_{PS}(x_1 \dots x_n) = P(x_1 \dots x_n). \quad (14.5)$$

Misalnya, perhatikan peristiwa yang dihasilkan sebelumnya: *[benar, salah, benar, benar]*. Probabilitas pengambilan sampel untuk peristiwa ini adalah

$$S_{PS}(\text{true}, \text{false}, \text{true}, \text{true}) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324.$$

Oleh karena itu, dalam batas N yang besar, kami mengharapkan 32,4% sampel merupakan peristiwa ini.

Setiap kali kami menggunakan persamaan perkiraan (" $\approx$ ") dalam hal berikut, kami mengartikannya dalam pengertian ini—bahwa probabilitas yang diestimasi menjadi tepat dalam batas sampel yang besar. Estimasi seperti itu disebut **konsisten**. Misalnya, seseorang dapat menghasilkan estimasi yang konsisten dari probabilitas setiap peristiwa yang ditentukan sebagian $x_1, \dots, x_m$, di mana $m \leq n$, sebagai berikut:

$$P(x_1 \dots x_m) \approx N_{PS}(x_1 \dots x_m) \quad (14.6)$$

Artinya, probabilitas kejadian dapat diestimasi sebagai fraksi dari semua kejadian lengkap yang dihasilkan oleh proses pengambilan sampel yang cocok dengan kejadian yang ditentukan sebagian. Misalnya, jika kita menghasilkan 1000 sampel dari jaringan sprinkler, dan 511 di antaranya memiliki $Rain = \text{true}$, maka estimasi probabilitas hujan, ditulis sebagai $P(Rain = \text{true})$, adalah 0,511.

Penolakan pengambilan sampel dalam jaringan Bayesian

Penolakan pengambilan sampel adalah metode umum untuk menghasilkan sampel dari distribusi yang sulit diambil sampelnya dengan distribusi yang mudah diambil sampelnya. Dalam bentuk yang paling sederhana, metode ini dapat digunakan untuk menghitung probabilitas bersyarat—yaitu, untuk menentukan $P(X | e)$. Algoritma REJECTION-SAMPLING ditunjukkan pada Gambar 5.14. Pertama, algoritma ini menghasilkan sampel dari distribusi sebelumnya yang ditentukan oleh jaringan. Kemudian, algoritma ini menolak semua sampel yang tidak cocok dengan bukti. Akhirnya, estimasi $P(X = x | e)$ diperoleh dengan menghitung seberapa sering $X = x$ muncul pada sampel yang tersisa.

Biarkan $P(X | e)$ menjadi distribusi yang diperkirakan yang dikembalikan oleh algoritma. Dari definisi algoritma, kita memiliki

$$\hat{\mathbf{P}}(X | \mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e}) = \frac{\mathbf{N}_{PS}(X, \mathbf{e})}{\mathbf{N}_{PS}(\mathbf{e})}$$

Dari Persamaan (5.6), hal ini menjadi

$$\hat{\mathbf{P}}(X | \mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X | \mathbf{e})$$

Ini berarti sampel penolakan menghasilkan estimasi yang konsisten dari probabilitas sebenarnya.

Melanjutkan contoh kita dari Gambar 5.12(a), mari kita asumsikan bahwa kita ingin memperkirakan $\mathbf{P}(\text{Rain} | \text{Sprinkler} = \text{true})$, menggunakan 100 sampel. Dari 100 sampel yang kita hasilkan, misalkan 73 sampel memiliki $\text{Sprinkler} = \text{false}$ dan ditolak, sementara 27 sampel memiliki $\text{Sprinkler} = \text{true}$; dari 27 sampel, 8 sampel memiliki $\text{Rain} = \text{true}$ dan 19 sampel memiliki $\text{Rain} = \text{false}$. Oleh karena itu,

$$\mathbf{P}(\text{Rain} | \text{Sprinkler} = \text{true}) \approx \text{NORMALIZE}(\langle 8, 19 \rangle) = \langle 0.296, 0.704 \rangle.$$

Jawaban yang benar adalah $\langle 0,3, 0,7 \rangle$. Semakin banyak sampel yang dikumpulkan, estimasi akan mendekati jawaban yang benar. Simpangan baku kesalahan dalam setiap probabilitas akan sebanding dengan $1/\sqrt{n}$, di mana n adalah jumlah sampel yang digunakan dalam estimasi.

Masalah terbesar dengan pengambilan sampel penolakan adalah ia menolak begitu banyak sampel! Fraksi sampel yang konsisten dengan bukti $\mathbf{e}$ turun secara eksponensial seiring dengan bertambahnya jumlah variabel bukti, sehingga prosedur tersebut tidak dapat digunakan untuk masalah yang kompleks.

Perhatikan bahwa pengambilan sampel penolakan sangat mirip dengan estimasi probabilitas bersyarat langsung dari dunia nyata. Misalnya, untuk memperkirakan $\mathbf{P}(\text{Rain} | \text{RedSkyAtNight} = \text{true})$, seseorang dapat menghitung seberapa sering hujan turun setelah langit merah diamati pada malam sebelumnya—mengabaikan malam-malam ketika langit tidak merah. (Di sini, dunia itu sendiri memainkan peran algoritma pembuatan sampel.) Jelas, ini bisa memakan waktu lama jika langit sangat jarang berwarna merah, dan itulah kelemahan pengambilan sampel penolakan.

Pembobotan kemungkinan

Pembobotan kemungkinan menghindari inefisiensi pengambilan sampel penolakan dengan hanya menghasilkan peristiwa yang konsisten dengan bukti $\mathbf{e}$. Ini adalah contoh khusus dari teknik statistik umum pengambilan **sampel penting**, yang disesuaikan untuk inferensi dalam jaringan Bayesian. Kami mulai dengan menjelaskan cara kerja algoritme; kemudian kami menunjukkan bahwa algoritme tersebut bekerja dengan benar—yaitu, menghasilkan estimasi probabilitas yang konsisten.

```

function REJECTION – SAMPLING( $X, e, bn, N$ ) returns an estimate of  $P(X|e)$ 
  inputs:  $X$ , the query variable
            $e$ , observed values for variables  $E$ 
            $bn$ , a Bayesian network
            $N$ , the total number of samples to be generated
  local variables:  $N$ , a vector of counts for each value of  $X$ , initially zero

  for  $j = 1$  to  $N$  do
     $x \leftarrow \text{PRIOR – SAMPLE}(bn)$ 
    if  $x$  is consistent with  $e$  then
       $N[x] \leftarrow N[x] + 1$  where  $x$  is the value of  $X$  in  $x$ 
  return NORMALIZE( $N$ )

```

Gambar 5.14 Algoritma penolakan-pengambilan sampel untuk menjawab pertanyaan yang diberikan bukti dalam jaringan Bayesian.

PEMBOBATAN KEMUNGKINAN (lihat Gambar 5.15) menetapkan nilai untuk variabel bukti E dan hanya mengambil sampel variabel nonbukti. Ini menjamin bahwa setiap peristiwa yang dihasilkan konsisten dengan bukti. Namun, tidak semua peristiwa sama. Sebelum menghitung jumlah dalam distribusi untuk variabel kueri, setiap peristiwa dibobot berdasarkan *kemungkinan* bahwa peristiwa tersebut sesuai dengan bukti, sebagaimana diukur dengan produk probabilitas bersyarat untuk setiap variabel bukti, yang diberikan induknya. Secara intuitif, peristiwa di mana bukti aktual tampak tidak mungkin harus diberi bobot yang lebih rendah.

Mari kita terapkan algoritma tersebut ke jaringan yang ditunjukkan pada Gambar 5.12(a), dengan kueri $P(Rain | Cloudy = true, WetGrass = true)$ dan urutan *Cloudy, Sprinkler, Rain, WetGrass*. (Urutan topologi apa pun bisa digunakan.) Prosesnya berjalan sebagai berikut: Pertama, bobot w ditetapkan menjadi 1,0. Kemudian, sebuah peristiwa dihasilkan:

1. *Cloudy* adalah variabel bukti dengan nilai *true*. Oleh karena itu, kita tetapkan

$$w \leftarrow w \times P(Cloudy = true) = 0.5.$$

2. *Sprinkler* bukan variabel bukti, jadi sampel dari $P(Sprinkler | Cloudy = true) = \langle 0.1, 0.9 \rangle$; misalkan ini mengembalikan *false*.
3. Demikian pula, sampel dari $P(Rain | Cloudy = true) = \langle 0.8, 0.2 \rangle$; misalkan ini mengembalikan *true*.
4. *WetGrass* adalah variabel bukti dengan nilai *true*. Oleh karena itu, kami menetapkan

$$w \leftarrow w \times P(WetGrass = true | Sprinkler = false, Rain = true) = 0.45.$$

Di sini WEIGHTED-SAMPLE mengembalikan kejadian $[true, false, true, true]$ dengan bobot 0,45, dan ini dihitung di bawah $Rain = true$.

```

function LIKELIHOOD – WEIGHTING( $X, e, bn, N$ ) returns an estimate of  $P(X|e)$ 
  inputs:  $X$ , the query variable
            $e$ , observed values for variables  $E$ 
            $bn$ , a Bayesian network specifying joint distribution  $P(X_1, \dots, X_n)$ 
            $N$ , the total number of samples to be generated
  local variables:  $W$ , a vector of weighted counts for each value of  $X$ , initially zero

  for  $j = 1$  to  $N$  do
     $x, w \leftarrow \text{WEIGHTED – SAMPLE}(bn, e)$ 
     $W[x] \leftarrow W[x] + w$  where  $x$  is the value of  $X$  in  $x$ 
  return NORMALIZE( $W$ )

function WEIGHTED – SAMPLE( $bn, e$ ) returns an event and a weight

   $w \leftarrow 1$ ;  $x \leftarrow$  an event with  $n$  elements initialized from  $e$ 
  foreach variable  $X_i$  in  $X_1, \dots, X_n$  do
    if  $X_i$  is an evidence variable with value  $x_i$  in  $e$ 
      then  $w \leftarrow w \times P(X_i = x_i | \text{parents}(X_i))$ 
      else  $x[i] \leftarrow$  a random sample from  $P(X_i | \text{parents}(X_i))$ 
  return  $x, w$ 

```

Gambar 5.15 Algoritma pembobotan kemungkinan untuk inferensi dalam jaringan Bayesian. Dalam WEIGHTED-SAMPLE, setiap variabel non-bukti diambil sampelnya menurut distribusi kondisional yang diberikan nilai-nilai yang telah diambil sampelnya untuk induk variabel, sementara bobot diakumulasikan berdasarkan kemungkinan untuk setiap variabel bukti.

Untuk memahami mengapa pembobotan kemungkinan berhasil, kita mulai dengan memeriksa probabilitas pengambilan sampel S_{WS} untuk WEIGHTED-SAMPLE. Ingat bahwa variabel bukti E ditetapkan dengan nilai e . Kita menyebut variabel non-bukti Z (termasuk variabel kueri X). Algoritme mengambil sampel setiap variabel di Z berdasarkan nilai induknya:

$$S_{WS}(Z, e) = \prod_{i=1}^l P(z_i | \text{parents}(z_i)) \quad (14.7)$$

Perhatikan bahwa $\text{Parents}(Z_i)$ dapat mencakup variabel non-bukti dan variabel bukti. Tidak seperti distribusi sebelumnya $P(z)$, distribusi S_{WS} memberikan perhatian pada bukti: nilai sampel untuk setiap Z_i akan dipengaruhi oleh bukti di antara leluhur Z_i . Misalnya, saat mengambil sampel *Sprinkler*, algoritme memberikan perhatian pada bukti $Cloudy = true$ dalam variabel induknya. Di sisi lain, S_{WS} kurang memperhatikan bukti daripada distribusi posterior true $P(z | e)$, karena nilai sampel untuk setiap Z_i mengabaikan bukti di antara non-leluhur Z_i . Misalnya, saat mengambil sampel *Sprinkler* dan *Rain*, algoritme mengabaikan bukti dalam variabel anak $WetGrass = true$; ini berarti akan menghasilkan banyak sampel

dengan $Sprinkler = false$ dan $Rain = false$ meskipun faktanya bukti sebenarnya mengesampingkan kasus ini.

Bobot kemungkinan w menggantikan perbedaan antara distribusi sampel aktual dan yang diinginkan. Bobot untuk sampel tertentu $\mathbf{x}$, yang terdiri dari $\mathbf{z}$ dan $\mathbf{e}$, adalah hasil perkalian dari kemungkinan untuk setiap variabel bukti yang diberikan induknya (beberapa atau semua mungkin berada di antara Z_i s):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^m P(e_i | \text{parents}(E_i)) \quad (14.8)$$

Dengan mengalikan Persamaan (5.7) dan (14.8), kita melihat bahwa probabilitas *tertimbang* dari sebuah sampel mempunyai bentuk yang sangat mudah

$$\begin{aligned} S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) &= \prod_{i=1}^l P(z_i | \text{parents}(z_i)) \prod_{i=1}^m P(e_i | \text{parents}(E_i)) \\ &= P(\mathbf{z}, \mathbf{e}) \end{aligned} \quad (14.9)$$

karena kedua produk tersebut mencakup semua variabel dalam jaringan, sehingga memungkinkan kita untuk menggunakan Persamaan (5.2) untuk probabilitas gabungan.

Sekarang mudah untuk menunjukkan bahwa estimasi bobot kemungkinan konsisten. Untuk setiap nilai x tertentu dari X , estimasi probabilitas posterior dapat dihitung sebagai berikut:

$$\begin{aligned} \hat{p}(x | e) &= \frac{1}{N} \sum_y N_{WS}(x, y, e) w(x, y, e) && \text{from LIKELIHOOD-WEIGHTING} \\ &= \frac{1}{N} \sum_y S_{WS}(x, y, e) w(x, y, e) && \text{for large } N \\ &= \frac{1}{N} \sum_y P(x, y, e) \\ &= \alpha P(x, e) = P(x | e). \end{aligned}$$

Oleh karena itu, pembobotan kemungkinan menghasilkan estimasi yang konsisten.

Karena pembobotan kemungkinan menggunakan semua sampel yang dihasilkan, pembobotan ini bisa jauh lebih efisien daripada pengambilan sampel penolakan. Namun, pembobotan ini akan mengalami penurunan kinerja seiring bertambahnya jumlah variabel bukti. Hal ini karena sebagian besar sampel akan memiliki bobot yang sangat rendah dan karenanya estimasi terbobot akan didominasi oleh sebagian kecil sampel yang memberikan kemungkinan yang lebih besar daripada yang sangat kecil terhadap bukti. Masalah ini diperburuk jika variabel bukti muncul di akhir pengurutan variabel, karena variabel nonbukti tidak akan memiliki bukti pada induk dan leluhurnya untuk memandu pembuatan sampel. Ini berarti sampel akan menjadi simulasi yang sedikit mirip dengan realitas yang disarankan oleh bukti.

Inferensi dengan simulasi rantai Markov

Algoritma **Markov chain Monte Carlo** (MCMC) bekerja sangat berbeda dari pengambilan sampel penolakan dan pembobotan kemungkinan. Alih-alih menghasilkan setiap sampel dari awal, algoritma MCMC menghasilkan setiap sampel dengan membuat perubahan acak pada sampel sebelumnya. Oleh karena itu, akan sangat membantu jika kita menganggap algoritma MCMC berada dalam status terkini tertentu yang menetapkan nilai untuk setiap variabel dan menghasilkan status berikutnya dengan melakukan perubahan acak pada status terkini. Di sini kami menjelaskan bentuk khusus MCMC yang disebut pengambilan sampel Gibbs, yang sangat cocok untuk jaringan Bayesian. (Bentuk lain, beberapa di antaranya jauh lebih canggih, dibahas dalam catatan di akhir bab.) Pertama-tama kami akan menjelaskan apa yang dilakukan algoritma tersebut, lalu kami akan menjelaskan mengapa ia bekerja.

Pengambilan sampel Gibbs dalam jaringan Bayesian+

Algoritma pengambilan sampel Gibbs untuk jaringan Bayesian dimulai dengan status arbitrer (dengan variabel bukti ditetapkan pada nilai yang diamati) dan menghasilkan status berikutnya dengan mengambil sampel secara acak nilai untuk salah satu variabel nonbukti X_i . Pengambilan sampel untuk X_i dilakukan dengan syarat *nilai variabel terkini dalam selimut Markov* X_i . Oleh karena itu, algoritme tersebut bergerak secara acak di sekitar ruang status—ruang kemungkinan penugasan lengkap—membalik satu variabel pada satu waktu, tetapi menjaga variabel bukti tetap.

Pertimbangkan kueri $\mathbf{P}(\text{Rain} \mid \text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$ yang diterapkan ke jaringan pada Gambar 5.12(a). Variabel bukti *Sprinkler* dan *WetGrass* ditetapkan pada nilai yang diamati dan variabel non-bukti *Cloudy* dan *Rain* diinisialisasi secara acak—misalkan *true* dan *false* secara berurutan. Jadi, status awal adalah $[\text{true}, \text{true}, \text{false}, \text{true}]$. Sekarang variabel non-bukti diambil sampelnya berulang kali dalam urutan yang sembarangan. Misalnya:

1. *Cloudy* diambil sampelnya, dengan nilai variabel selimut Markov saat ini: dalam kasus ini, kami mengambil sampel dari $\mathbf{P}(\text{Cloudy} \mid \text{Sprinkler} = \text{true}, \text{Rain} = \text{false})$. (Sebentar lagi, kami akan menunjukkan cara menghitung distribusi ini.) Misalkan hasilnya adalah *Cloudy* = *false*. Maka status terkini yang baru adalah $[\text{false}, \text{true}, \text{false}, \text{true}]$.
2. *Rain* diambil sampelnya, dengan nilai terkini dari variabel selimut Markov-nya: dalam kasus ini, kami mengambil sampel dari $\mathbf{P}(\text{Rain} \mid \text{Cloudy} = \text{false}, \text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$. Misalkan ini menghasilkan *Rain* = *true*. Status terkini yang baru adalah $[\text{false}, \text{true}, \text{true}, \text{true}]$.

Setiap status yang dikunjungi selama proses ini adalah sampel yang berkontribusi pada estimasi untuk variabel kueri *rain*. Jika proses mengunjungi 20 status di mana *rain* benar dan 60 status di mana Hujan salah, maka jawaban untuk kueri tersebut adalah $\text{NORMALIZE}(\langle 20, 60 \rangle) = \langle 0,25, 0,75 \rangle$. Algoritme lengkapnya ditunjukkan pada Gambar 5.16.

Mengapa pengambilan sampel Gibbs berhasil

Kami sekarang akan menunjukkan bahwa pengambilan sampel Gibbs menghasilkan estimasi yang konsisten untuk probabilitas posterior. Materi dalam bagian ini cukup teknis, tetapi klaim dasarnya lugas: proses pengambilan sampel mencapai "*keseimbangan dinamis*" di mana *fraksi waktu jangka panjang yang dihabiskan di setiap keadaan berbanding lurus dengan probabilitas posteriornya*. Properti yang luar biasa ini mengikuti **probabilitas transisi** spesifik yang dengannya proses bergerak dari satu keadaan ke keadaan lain, sebagaimana didefinisikan oleh distribusi kondisional yang diberikan selimut Markov dari variabel yang diambil sampelnya.

```

function GIBBS – ASK( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $\mathbf{P}(X|\mathbf{e})$ 
  local variables:  $\mathbf{N}$ , a vector of counts for each value of  $X$ , initially zero
                     $\mathbf{Z}$ , the nonevidence variables in  $bn$ 
                     $\mathbf{x}$ , the current state of the network, initially copied from  $\mathbf{e}$ 

  initialize  $\mathbf{x}$  with random values for the variables in  $\mathbf{Z}$ 
  for  $j = 1$  to  $N$  do
    for each  $Z_i$  in  $\mathbf{Z}$  do
      set the value of  $Z_i$  in  $\mathbf{x}$  by sampling from  $\mathbf{P}(Z_i|mb(Z_i))$ 
       $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$  where  $x$  is the value of  $X$  in  $\mathbf{x}$ 
  return NORMALIZE( $\mathbf{N}$ )
  
```

Gambar 5.16 Algoritma pengambilan sampel Gibbs untuk inferensi perkiraan dalam jaringan Bayesian; versi ini berputar di antara variabel, tetapi memilih variabel secara acak juga berfungsi.

Misalkan $q(\mathbf{x} \rightarrow \mathbf{x}')$ adalah probabilitas bahwa proses tersebut membuat transisi dari keadaan $\mathbf{x}$ ke keadaan $\mathbf{x}'$. Probabilitas transisi ini mendefinisikan apa yang disebut **rantai Markov** pada ruang keadaan. Sekarang anggaplah kita menjalankan rantai Markov selama t langkah, dan misalkan $\pi_t(\mathbf{x})$ adalah probabilitas bahwa sistem berada dalam keadaan $\mathbf{x}$ pada waktu t . Demikian pula, misalkan $\pi_{t+1}(\mathbf{x}')$ adalah probabilitas berada dalam keadaan $\mathbf{x}'$ pada waktu $t + 1$. Diberikan $\pi_t(\mathbf{x})$, kita dapat menghitung $\pi_{t+1}(\mathbf{x}')$ dengan menjumlahkan, untuk semua keadaan yang dapat dialami sistem pada waktu t , probabilitas berada dalam keadaan tersebut dikalikan probabilitas melakukan transisi ke $\mathbf{x}'$:

$$\pi_{t+1}(\mathbf{x}') = \sum_x \pi_t(x) q(x \rightarrow x')$$

Kita katakan bahwa rantai telah mencapai **distribusi stasionernya** jika $\pi_t = \pi_{t+1}$. Mari kita sebut distribusi stasioner ini π ; oleh karena itu persamaan yang mendefinisikannya adalah

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \quad \text{for all } \mathbf{x}'. \quad (14.10)$$

Asalkan distribusi probabilitas transisi q bersifat **ergodis**—yaitu, setiap keadaan dapat dicapai dari setiap keadaan lain dan tidak ada siklus periodik yang ketat—ada tepat satu distribusi π yang memenuhi persamaan ini untuk setiap q yang diberikan.

Persamaan (5.10) dapat dibaca dengan mengatakan bahwa “arus keluar” yang diharapkan dari setiap keadaan (yaitu, “populasi” saat ini) sama dengan “arus masuk” yang diharapkan dari semua keadaan. Salah satu cara yang jelas untuk memenuhi hubungan ini adalah jika aliran yang diharapkan antara setiap pasangan keadaan sama di kedua arah; yaitu,

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \quad \text{for all } \mathbf{x}, \mathbf{x}'. \quad (14.11)$$

Bila persamaan ini berlaku, kita katakan bahwa $q(\mathbf{x} \rightarrow \mathbf{x}')$ berada dalam **keseimbangan** terperinci dengan $\pi(\mathbf{x})$.

Kita dapat menunjukkan bahwa keseimbangan terperinci menyiratkan stasioneritas hanya dengan menjumlahkan $\mathbf{x}$ dalam Persamaan (5.11). Kita memiliki

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

di mana langkah terakhir mengikuti karena transisi dari $\mathbf{x}'$ dijamin akan terjadi.

Probabilitas transisi $q(\mathbf{x} \rightarrow \mathbf{x}')$ yang didefinisikan oleh langkah pengambilan sampel dalam GIBBS-ASK sebenarnya adalah kasus khusus dari definisi pengambilan sampel Gibbs yang lebih umum, yang menurutnya setiap variabel diambil sampelnya secara kondisional berdasarkan nilai saat ini dari *semua* variabel lainnya. Kita mulai dengan menunjukkan bahwa definisi umum pengambilan sampel Gibbs ini memenuhi persamaan keseimbangan terperinci dengan distribusi stasioner yang sama dengan $P(\mathbf{x}|\mathbf{e})$, (distribusi posterior sebenarnya pada variabel non-bukti). Kemudian, kita cukup mengamati bahwa, untuk jaringan Bayesian, pengambilan sampel secara kondisional pada semua variabel setara dengan pengambilan sampel secara kondisional pada selimut Markov variabel.

Untuk menganalisis pengambil sampel Gibbs umum, yang mengambil sampel setiap X_i secara bergantian dengan probabilitas transisi q_i yang dikondisikan pada semua variabel lainnya, kita mendefinisikan $\bar{\mathbf{X}}_i$ sebagai variabel-variabel lain ini (kecuali variabel bukti); nilai-nilai mereka dalam keadaan saat ini adalah $\bar{\mathbf{x}}_i$. Jika kita mengambil sampel nilai baru x'_i untuk X_i secara kondisional pada semua variabel lainnya, termasuk bukti, kita memiliki

$$q_i(\mathbf{x} \rightarrow \mathbf{x}') = q_i((x_i, \bar{\mathbf{x}}_i) \rightarrow (x'_i, \bar{\mathbf{x}}_i)) = P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e})$$

Sekarang kami tunjukkan bahwa probabilitas transisi untuk setiap langkah sampler Gibbs seimbang secara detail dengan posterior sebenarnya:

$$\begin{aligned}
\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') &= P(\mathbf{x} | \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \\
&= P(x_i, \bar{\mathbf{x}}_i | \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) \\
&= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(\bar{\mathbf{x}}_i | \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) && \text{(menggunakan aturan rantai pada suku pertama)} \\
&= P(x_i | \bar{\mathbf{x}}_i, \mathbf{e}) P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) && \text{(menggunakan aturan rantai terbalik)} \\
&= \pi(\mathbf{x}') q_i(\mathbf{x}' \rightarrow \mathbf{x}).
\end{aligned}$$

Kita dapat menganggap loop “**untuk setiap** Z_i dalam **Z do**” pada Gambar 5.16 sebagai pendefinisian satu probabilitas transisi besar q yang merupakan komposisi sekuensial $q_1 \circ q_2 \circ \dots \circ q_n$ dari probabilitas transisi untuk variabel individual. Mudah untuk menunjukkan bahwa jika masing-masing q_i dan q_j memiliki π sebagai distribusi stasionernya, maka komposisi sekuensial $q_i \circ q_j$ juga demikian; maka probabilitas transisi q untuk seluruh loop memiliki $P(\mathbf{x} | \mathbf{e})$ sebagai distribusi stasionernya. Akhirnya, kecuali CPT berisi probabilitas 0 atau 1—yang dapat menyebabkan ruang keadaan menjadi terputus—mudah untuk melihat bahwa q bersifat ergodis. Oleh karena itu, sampel yang dihasilkan oleh pengambilan sampel Gibbs pada akhirnya akan diambil dari distribusi posterior sebenarnya.

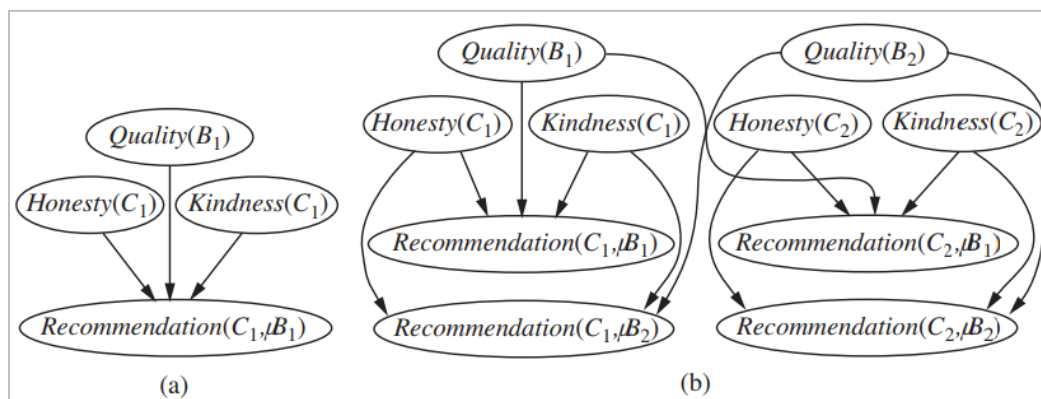
Langkah terakhir adalah menunjukkan cara melakukan langkah pengambilan sampel Gibbs umum—mengambil sampel X_i dari $\mathbf{P}(X'_i | \bar{\mathbf{x}}_i, \mathbf{e})$ —dalam jaringan Bayesian. Ingat kembali dari halaman sebelumnya bahwa sebuah variabel tidak bergantung pada semua variabel lain yang diberikan selimut Markov-nya; oleh karena itu,

$$P(x'_i | \bar{\mathbf{x}}_i, \mathbf{e}) = P(x'_i | mb(X_i)),$$

di mana $mb(X_i)$ menunjukkan nilai variabel dalam selimut Markov X_i , $MB(X_i)$. Probabilitas variabel yang diberikan selimut Markov-nya sebanding dengan probabilitas variabel yang diberikan orang tuanya dikalikan probabilitas setiap anak yang diberikan orang tuanya masing-masing:

$$\begin{aligned}
P(x'_i | mb(X_i)) \\
= \alpha P(x'_i | \text{parents}(X_i)) \times \prod_{Y_j \in \text{Children}(X_i)} P(y_j | \text{parents}(Y_j)). \quad (14.12)
\end{aligned}$$

Oleh karena itu, untuk membalik setiap variabel X_i yang dikondisikan pada selimut Markov-nya, jumlah perkalian yang diperlukan sama dengan jumlah anak X_i .



Gambar 5.17 (a) Bayes net untuk satu pelanggan C_1 yang merekomendasikan satu buku B_1 . *Honest* (C_1) adalah Boolean, sedangkan variabel lainnya memiliki nilai integer dari 1 hingga 5. (b) Bayes net dengan dua pelanggan dan dua buku.

5.6 MODEL KEMUNGKINAN RELASIONAL DAN ORDE PERTAMA

Representasional yang dimiliki oleh logika orde pertama dibandingkan dengan logika proposisional. Logika orde pertama berkomitmen pada keberadaan objek dan hubungan di antara objek tersebut dan dapat mengungkapkan fakta tentang beberapa atau semua objek dalam suatu domain. Hal ini sering kali menghasilkan representasi yang jauh lebih ringkas daripada deskripsi proposisional yang setara. Sekarang, jaringan Bayesian pada dasarnya bersifat proposisional: himpunan variabel acak bersifat tetap dan terbatas, dan masing-masing memiliki domain nilai yang mungkin tetap. Fakta ini membatasi penerapan jaringan Bayesian. *Jika kita dapat menemukan cara untuk menggabungkan teori probabilitas dengan kekuatan ekspresif representasi orde pertama, kita berharap dapat meningkatkan secara dramatis rentang masalah yang dapat ditangani.*

Misalnya, anggaplah bahwa pengecer buku daring ingin memberikan evaluasi keseluruhan produk berdasarkan rekomendasi yang diterima dari pelanggannya. Evaluasi akan mengambil bentuk distribusi posterior atas kualitas buku, berdasarkan bukti yang tersedia. Solusi paling sederhana adalah mendasarkan evaluasi pada rekomendasi rata-rata, mungkin dengan varians yang ditentukan oleh jumlah rekomendasi, tetapi ini gagal memperhitungkan fakta bahwa beberapa pelanggan lebih baik daripada yang lain dan beberapa kurang jujur daripada yang lain. Pelanggan yang baik cenderung memberikan rekomendasi yang tinggi bahkan untuk buku yang cukup biasa-biasa saja, sementara pelanggan yang tidak jujur memberikan rekomendasi yang sangat tinggi atau sangat rendah karena alasan selain kualitas—misalnya, mereka mungkin bekerja untuk penerbit.

Untuk satu pelanggan C_1 , yang merekomendasikan satu buku B_1 , jaringan Bayes mungkin terlihat seperti yang ditunjukkan pada Gambar 5.17(a), ekspresi dengan tanda kurung seperti *Honest*(C_1) hanyalah simbol-simbol yang rumit—dalam hal ini, nama-nama yang rumit untuk variabel acak.) Dengan dua pelanggan dan dua buku, jaringan Bayes tampak seperti yang ada pada Gambar 5.17(b). Untuk jumlah buku dan pelanggan yang lebih besar, sama sekali tidak praktis untuk menentukan jaringan secara manual.

Untungnya, jaringan tersebut memiliki banyak struktur yang berulang. Setiap variabel *Recommendation* (c, b) memiliki induknya, yaitu variabel *Honest* (c), *Kindness* (c), dan *Quality* (b). Selain itu, CPT untuk semua variabel *Recommendation* (c, b) identik, begitu pula untuk semua variabel *Honest* (c), dan seterusnya. Situasinya tampaknya dibuat khusus untuk bahasa tingkat pertama. Kami ingin mengatakan sesuatu seperti

$$Recommendation(c, b) \sim RecCPT(Honest(c), Kindness(c), Quality(b))$$

dengan maksud yang dimaksudkan bahwa rekomendasi pelanggan untuk sebuah buku bergantung pada kejujuran dan kebaikan pelanggan dan kualitas buku menurut beberapa CPT tetap. Bagian ini mengembangkan bahasa yang memungkinkan kita mengatakan hal ini secara tepat, dan lebih banyak lagi.

Kemungkinan dunia

Ingat dari Bab 4 bahwa model probabilitas mendefinisikan satu set Ω dari kemungkinan dunia dengan probabilitas $P(\omega)$ untuk setiap dunia ω . Untuk jaringan Bayesian, kemungkinan dunia adalah penugasan nilai ke variabel; untuk kasus Boolean khususnya, kemungkinan dunia identik dengan logika proposisional. Untuk model probabilitas orde pertama, tampaknya kita memerlukan kemungkinan dunia untuk menjadi logika orde pertama—yaitu, satu set objek dengan hubungan di antara mereka dan interpretasi yang memetakan simbol konstan ke objek, simbol predikat ke hubungan, dan simbol fungsi ke fungsi pada objek tersebut. Model tersebut juga perlu menentukan probabilitas untuk setiap kemungkinan dunia tersebut, seperti halnya jaringan Bayesian yang menentukan probabilitas untuk setiap penugasan nilai pada variabel.

Mari kita asumsikan, sejenak, bahwa kita telah menemukan cara untuk melakukan ini. Kemudian, seperti biasa, kita dapat memperoleh probabilitas kalimat logis orde pertama ϕ sebagai penjumlahan atas kemungkinan dunia yang benar:

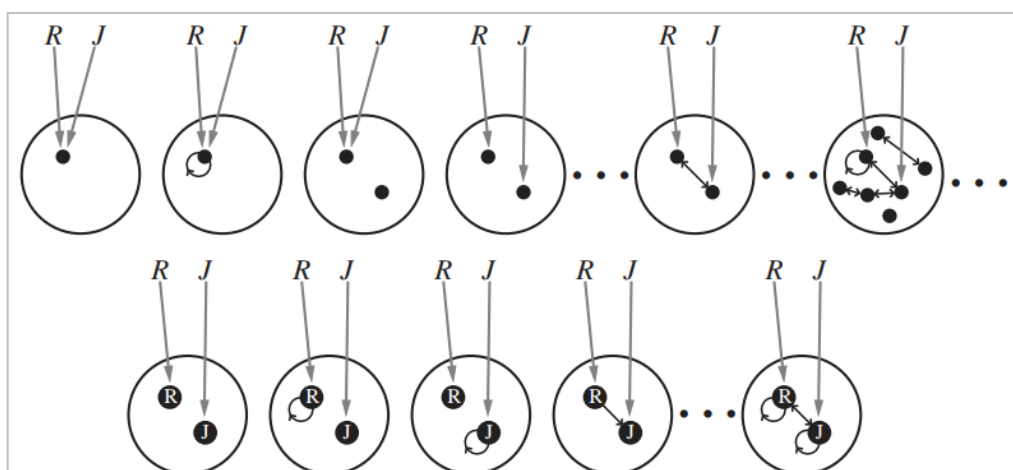
$$P(\phi) = \sum_{\omega: \phi \text{ is true in } \omega} P(\omega). \quad (14.13)$$

Probabilitas bersyarat $P(\phi|\mathbf{e})$ dapat diperoleh dengan cara yang sama, jadi pada prinsipnya kita dapat mengajukan pertanyaan apa pun yang kita inginkan dari model kita—misalnya, "Buku mana yang kemungkinan besar akan sangat direkomendasikan oleh pelanggan yang tidak jujur?"—dan mendapatkan jawaban. Sejauh ini, semuanya baik-baik saja.

Namun, ada masalah: himpunan model orde pertama tidak terbatas. Kita melihat ini secara eksplisit yang kita tunjukkan lagi pada Gambar 5.18 (atas). Ini berarti bahwa (1) penjumlahan dalam Persamaan (5.13) mungkin tidak layak, dan (2) menentukan distribusi yang lengkap dan konsisten pada himpunan dunia yang tak terbatas bisa jadi sangat sulit.

Ketika asumsi dasar semantik basis data gagal berlaku, RPM tidak akan bekerja dengan baik. Misalnya, pengecer buku mungkin menggunakan ISBN (Nomor Buku Standar Internasional) sebagai simbol konstan untuk memberi nama setiap buku, meskipun buku "logis" tertentu (misalnya, "Gone With the Wind") mungkin memiliki beberapa ISBN. Masuk

akal untuk menggabungkan rekomendasi di beberapa ISBN, tetapi pengecer mungkin tidak tahu pasti ISBN mana yang benar-benar buku yang sama. (Perhatikan bahwa kami tidak mereifikasi *salinan* buku secara *individual*, yang mungkin diperlukan untuk penjualan buku bekas, penjualan mobil, dan sebagainya.) Lebih buruk lagi, setiap pelanggan diidentifikasi oleh ID login, tetapi pelanggan yang tidak jujur mungkin memiliki ribuan ID! Dalam bidang keamanan komputer, beberapa ID ini disebut **sibyl** dan penggunaannya untuk mengacaukan sistem reputasi disebut **serangan sibyl**. Jadi, bahkan aplikasi sederhana dalam domain online yang relatif terdefinisi dengan baik melibatkan **ketidakpastian keberadaan** (apa buku dan pelanggan sebenarnya yang mendasari data yang diamati) dan **ketidakpastian identitas** (simbol mana yang benar-benar merujuk ke objek yang sama). Kita perlu mengambil keputusan dan mendefinisikan model probabilitas berdasarkan semantik standar logika orde pertama, yang mana kemungkinan dunia bervariasi dalam objek yang dikandungnya dan dalam pemetaan dari simbol ke objek.



Gambar 5.18 Atas: Beberapa anggota himpunan semua kemungkinan dunia untuk suatu bahasa dengan dua simbol konstan, R dan J , dan satu simbol relasi biner, di bawah semantik standar untuk logika orde pertama. Bawah: kemungkinan dunia di bawah semantik basis data. Penafsiran simbol konstan bersifat tetap, dan ada objek yang berbeda untuk setiap simbol konstan.

Model probabilitas relasional

Seperti logika orde pertama, RPM memiliki simbol konstanta, fungsi, dan predikat. (Ternyata lebih mudah untuk melihat predikat sebagai fungsi yang mengembalikan *true* atau *false*.) Kami juga akan mengasumsikan tanda **type signature** untuk setiap fungsi, yaitu, spesifikasi tipe setiap argumen dan nilai fungsi. Jika tipe setiap objek diketahui, banyak kemungkinan dunia palsu yang dihilangkan oleh mekanisme ini. Untuk domain rekomendasi buku, tipenya adalah *Customer* dan *Book*, dan tanda tangan tipe untuk fungsi dan predikat adalah sebagai berikut:

$$\text{Honest: Customer} \rightarrow \text{true, false} \quad \text{Kindness: Customer} \rightarrow 1,2,3,4,5$$

Quality: Book $\rightarrow 1,2,3,4,5$

Recommendation : *Customer* $\times$ *Book* $\rightarrow \{1,2,3,4,5\}$

Simbol konstanta akan berupa nama pelanggan dan buku apa pun yang muncul dalam set data pengecer. Dalam contoh yang diberikan sebelumnya (Gambar 5.17(b)), ini adalah C_1 , C_2 dan B_1 , B_2 .

Dengan konstanta dan tipenya, bersama dengan fungsi dan tanda tangan tipenya, variabel acak RPM diperoleh dengan membuat instance setiap fungsi dengan setiap kemungkinan kombinasi objek: $Honest(C_1)$, $Quality(B_2)$, $Recommendation(C_1, B_2)$, dan seterusnya. Ini persis variabel yang muncul pada Gambar 5.17(b). Karena setiap tipe hanya memiliki jumlah instance yang terbatas, jumlah variabel acak dasar juga terbatas.

Untuk melengkapi RPM, kita harus menulis dependensi yang mengatur variabel acak ini. Ada satu pernyataan dependensi untuk setiap fungsi, di mana setiap argumen fungsi adalah variabel logis (yaitu, variabel yang berkisar pada objek, seperti dalam logika orde pertama):

$Honest(c) \sim 0.99, 0.01$

$Kindness(c) \sim 0.1, 0.1, 0.2, 0.3, 0.3$

$Quality(b) \sim 0.05, 0.2, 0.4, 0.2, 0.15$

$Recommendation(c, b) \sim RecCPT(Honest(c), Kindness(c), Quality(b))$

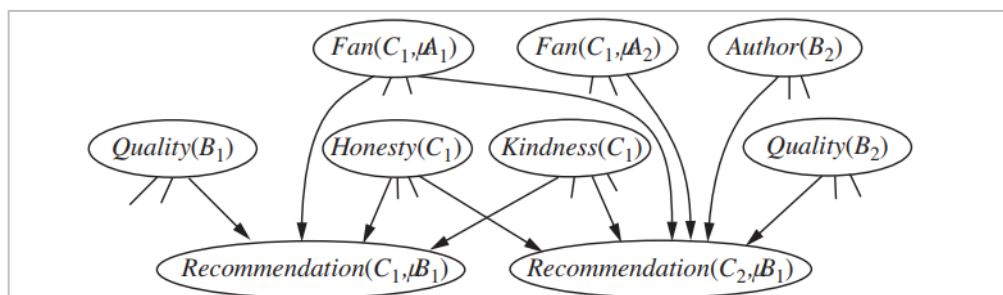
di mana $RecCPT$ adalah distribusi kondisional yang didefinisikan secara terpisah dengan $2 \times 5 \times 5 = 50$ baris, masing-masing dengan 5 entri. Semantik RPM dapat diperoleh dengan membuat contoh dependensi ini untuk semua konstanta yang diketahui, yang memberikan jaringan Bayesian (seperti pada Gambar 5.17(b)) yang mendefinisikan distribusi gabungan atas variabel acak RPM.⁸

Kita dapat menyempurnakan model dengan memperkenalkan **independensi khusus konteks** untuk mencerminkan fakta bahwa pelanggan yang tidak jujur mengabaikan kualitas saat memberikan rekomendasi; lebih jauh lagi, kebaikan tidak berperan dalam keputusan mereka. Independensi khusus konteks memungkinkan variabel menjadi independen dari beberapa induknya dengan nilai tertentu dari yang lain; dengan demikian, $Recommendation(c, b)$ independen dari $Kindness(c)$ dan $Quality(b)$ saat $Honest(c) = false$:

$Recommendation(c, b) \sim \text{if } Honest(c) \text{ then}$

$HonestRecCPT(Kindness(c), Quality(b))$

else $\langle 0.4, 0.1, 0.0, 0.1, 0.4 \rangle$.



Gambar 5.19 Fragmen jaringan Bayes yang ekuivalen ketika Penulis (B_2) tidak diketahui.

Ketergantungan semacam ini mungkin terlihat seperti pernyataan if–then–else biasa pada bahasa pemrograman, tetapi ada perbedaan utama: mesin inferensi tidak selalu mengetahui *nilai dari pengujian kondisional!*

Kita dapat menguraikan model ini dengan cara yang tak terbatas untuk membuatnya lebih realistis. Misalnya, anggaplah bahwa seorang pelanggan jujur yang merupakan penggemar penulis buku selalu memberi buku tersebut nilai 5, terlepas dari kualitasnya:

$Recommendation(c, b) \sim$ **if** $Honest(c)$ **then**
 if $Fan(c, Author(b))$ **then** $Exactly(5)$
 else $HonestRecCPT(Kindness(c), Quality(b))$
 else $\langle 0.4, 0.1, 0.0, 0.1, 0.4 \rangle$.

Sekali lagi, uji kondisional $Fan(c, Author(b))$ tidak diketahui, tetapi jika seorang pelanggan hanya memberikan angka 5 untuk buku-buku penulis tertentu dan tidak terlalu baik, maka probabilitas posterior bahwa pelanggan tersebut adalah penggemar penulis tersebut akan tinggi. Lebih jauh, distribusi posterior akan cenderung mengabaikan angka 5 pelanggan dalam mengevaluasi kualitas buku-buku penulis tersebut.

Dalam contoh sebelumnya, kami secara implisit berasumsi bahwa nilai $Author(b)$ diketahui untuk setiap b , tetapi ini mungkin tidak terjadi. Bagaimana sistem dapat bernalar tentang apakah, katakanlah, C_1 adalah penggemar $Author(B_2)$ ketika $Author(B_2)$ tidak diketahui? Jawabannya adalah bahwa sistem mungkin harus bernalar tentang semua kemungkinan *authors*. Misalkan (untuk menyederhanakannya) bahwa hanya ada dua penulis, A_1 dan A_2 . Maka $Author(B_2)$ adalah variabel acak dengan dua kemungkinan nilai, A_1 dan A_2 , dan merupakan induk dari $Recommendation(C_1, B_2)$. Variabel $Fan(C_1, A_1)$ dan $Fan(C_1, A_2)$ juga merupakan induk. Distribusi kondisional untuk $Recommendation(C_1, B_2)$ pada dasarnya merupakan **multiplexer** di mana induk $Author(B_2)$ bertindak sebagai penyeleksi untuk memilih $Fan(C_1, A_1)$ dan $Fan(C_1, A_2)$ mana yang benar-benar dapat memengaruhi rekomendasi. Sebagian dari jaringan Bayes yang ekuivalen ditunjukkan pada Gambar 5.19. Ketidakpastian dalam nilai $Author(B_2)$, yang memengaruhi struktur ketergantungan jaringan, merupakan contoh **ketidakpastian relasional**.

Jika Anda bertanya-tanya bagaimana sistem dapat menentukan siapa penulis B_2 : pertimbangkan kemungkinan bahwa tiga pelanggan lain adalah penggemar A_1 (dan tidak

memiliki penulis favorit yang sama) dan ketiganya telah memberi B_2 nilai 5, meskipun sebagian besar pelanggan lain menganggapnya cukup suram. Dalam kasus tersebut, sangat mungkin bahwa A_1 adalah penulis B_2 . Munculnya penalaran canggih seperti ini dari model RPM yang hanya terdiri dari beberapa baris merupakan contoh menarik tentang bagaimana pengaruh probabilistik menyebar melalui jaringan interkoneksi di antara objek dalam model. Karena semakin banyak dependensi dan objek yang ditambahkan, gambaran yang disampaikan oleh distribusi posterior sering kali menjadi semakin jelas.

Pertanyaan berikutnya adalah bagaimana melakukan inferensi dalam RPM. Salah satu pendekatannya adalah mengumpulkan bukti dan kueri serta simbol konstan di dalamnya, membangun jaringan Bayes yang setara, dan menerapkan salah satu metode inferensi yang dibahas dalam bab ini. Teknik ini disebut **unrolling**. Kelemahan yang jelas adalah bahwa jaringan Bayes yang dihasilkan mungkin sangat besar. Lebih jauh lagi, jika ada banyak objek kandidat untuk relasi atau fungsi yang tidak diketahui—misalnya, penulis B_2 yang tidak diketahui—maka beberapa variabel dalam jaringan mungkin memiliki banyak induk.

Untungnya, banyak yang dapat dilakukan untuk meningkatkan algoritme inferensi generik. Pertama, keberadaan substruktur berulang dalam jaringan Bayes yang tidak digulung berarti bahwa banyak faktor yang dibangun selama eliminasi variabel (dan jenis tabel serupa yang dibangun oleh algoritma pengelompokan) akan identik; skema caching yang efektif telah menghasilkan percepatan tiga kali lipat untuk jaringan besar. Kedua, metode inferensi yang dikembangkan untuk memanfaatkan independensi khusus konteks dalam jaringan Bayes menemukan banyak aplikasi dalam RPM. Ketiga, algoritma inferensi MCMC memiliki beberapa properti menarik ketika diterapkan pada RPM dengan ketidakpastian relasional. MCMC bekerja dengan mengambil sampel dari kemungkinan dunia yang lengkap, sehingga dalam setiap status struktur relasional diketahui sepenuhnya. Dalam contoh yang diberikan sebelumnya, setiap status MCMC akan menentukan nilai $Author(B_2)$, sehingga penulis potensial lainnya tidak lagi menjadi induk dari simpul rekomendasi untuk B_2 . Untuk MCMC, ketidakpastian relasional tidak menyebabkan peningkatan dalam kompleksitas jaringan; sebaliknya, proses MCMC mencakup transisi yang mengubah struktur relasional, dan karenanya struktur ketergantungan, dari jaringan yang tidak digulung.

Semua metode yang baru saja dijelaskan mengasumsikan bahwa RPM harus diurai sebagian atau seluruhnya menjadi jaringan Bayesian. Ini persis sama dengan metode **proposisionalisasi** untuk inferensi logis orde pertama. Pembukti teorema resolusi dan sistem pemrograman logika menghindari proposisionalisasi dengan membuat contoh variabel logis hanya sebagaimana diperlukan untuk membuat inferensi berjalan; yaitu, mereka *mengangkat* proses inferensi di atas tingkat kalimat proposisional dasar dan membuat setiap langkah yang diangkat melakukan pekerjaan dari banyak langkah dasar. Ide yang sama diterapkan dalam inferensi probabilistik. Misalnya, dalam algoritma eliminasi variabel, faktor yang diangkat dapat mewakili seluruh rangkaian faktor dasar yang menetapkan probabilitas pada variabel acak dalam RPM, di mana variabel acak tersebut hanya berbeda dalam simbol konstan yang digunakan untuk membangunnya. Rincian metode ini berada di luar cakupan buku ini, tetapi referensi diberikan di akhir bab.

Model probabilitas semesta terbuka

Sebelumnya kami berpendapat bahwa semantik basis data sesuai untuk situasi di mana kami mengetahui secara pasti kumpulan objek relevan yang ada dan dapat mengidentifikasinya dengan jelas. (Khususnya, semua pengamatan tentang suatu objek dikaitkan dengan simbol konstan yang menamainya.) Namun, dalam banyak situasi dunia nyata, asumsi ini tidak dapat dipertahankan. Kami memberikan contoh beberapa ISBN dan serangan sibyl dalam domain rekomendasi buku (yang akan kami bahas sebentar lagi), tetapi fenomena ini jauh lebih meluas:

- Sistem penglihatan tidak mengetahui apa yang ada, jika ada, di sekitar sudut berikutnya, dan mungkin tidak mengetahui apakah objek yang dilihatnya sekarang adalah objek yang sama yang dilihatnya beberapa menit yang lalu.
- Sistem pemahaman teks tidak mengetahui terlebih dahulu entitas yang akan ditampilkan dalam teks, dan harus bernalar tentang apakah frasa seperti "Mary," "Dr. Smith," "dia," "dokter jantungnya," "ibunya," dan seterusnya merujuk pada objek yang sama.
- Seorang analis intelijen yang memburu mata-mata tidak pernah tahu berapa banyak mata-mata yang sebenarnya ada dan hanya dapat menebak apakah berbagai nama samaran, nomor telepon, dan penampakan milik individu yang sama.

Faktanya, sebagian besar kognisi manusia tampaknya memerlukan pembelajaran tentang objek apa yang ada dan mampu menghubungkan pengamatan—yang hampir tidak pernah disertai dengan ID unik—dengan objek yang dihipotesiskan di dunia.

Karena alasan ini, kita perlu mampu menulis apa yang disebut model probabilitas **semesta terbuka** atau OUPM berdasarkan semantik standar logika orde pertama, seperti yang diilustrasikan di bagian atas Gambar 5.18. Bahasa untuk OUPM menyediakan cara menulis model tersebut dengan mudah sekaligus menjamin distribusi probabilitas yang unik dan konsisten pada ruang tak terbatas dari kemungkinan dunia.

Ide dasarnya adalah untuk memahami bagaimana jaringan Bayesian dan RPM biasa berhasil mendefinisikan model probabilitas yang unik dan mentransfer wawasan tersebut ke pengaturan orde pertama. Intinya, jaringan Bayes *menghasilkan* setiap kemungkinan dunia, peristiwa demi peristiwa, dalam urutan topologi yang ditentukan oleh struktur jaringan, di mana setiap peristiwa merupakan penugasan nilai ke variabel. RPM memperluas ini ke seluruh rangkaian peristiwa, yang ditentukan oleh kemungkinan perwujudan variabel logis dalam predikat atau fungsi tertentu. OUPM melangkah lebih jauh dengan memungkinkan langkah-langkah generatif yang *menambahkan objek* ke kemungkinan dunia yang sedang dibangun, di mana jumlah dan jenis objek dapat bergantung pada objek yang sudah ada di dunia tersebut. Artinya, peristiwa yang dihasilkan bukanlah penetapan nilai ke variabel, tetapi *keberadaan* objek itu sendiri.

Salah satu cara untuk melakukan ini dalam OUPM adalah dengan menambahkan pernyataan yang mendefinisikan distribusi bersyarat atas jumlah objek dari berbagai jenis. Misalnya, dalam domain rekomendasi buku, kita mungkin ingin membedakan antara

pelanggan (orang sungguhan) dan ID *login* mereka. Misalkan kita mengharapkan antara 100 dan 10.000 pelanggan berbeda (yang tidak dapat kita amati secara langsung). Kita dapat menyatakan ini sebagai distribusi log-normal sebelumnya sebagai berikut:

$$\#Customer \sim \text{LogNormal}[6.9, 2.33^2]().$$

Kami mengharapkan pelanggan yang jujur hanya memiliki satu ID, sedangkan pelanggan yang tidak jujur mungkin memiliki antara 10 hingga 1000 ID:

$$\#LoginID(Owner = c) \sim \text{if } Honest(c) \text{ then } Exactly(1) \\ \text{else } \text{LogNormal}[6.9, 2.3^2]().$$

Pernyataan ini mendefinisikan jumlah ID login untuk pemilik tertentu, yang merupakan pelanggan. Fungsi *Pemilik* disebut **fungsi asal** karena menyatakan dari mana setiap objek yang dihasilkan berasal. Dalam semantik formal BLOG (berbeda dari logika orde pertama), elemen domain di setiap dunia yang mungkin sebenarnya adalah riwayat pembuatan (misalnya, "ID login keempat dari pelanggan ketujuh") dan bukan token sederhana.

Tunduk pada kondisi teknis asiklialitas dan dasar yang kuat yang serupa dengan RPM, model semesta terbuka semacam ini mendefinisikan distribusi unik atas kemungkinan dunia. Lebih jauh, ada algoritma inferensi sedemikian rupa sehingga, untuk setiap model yang terdefinisi dengan baik dan setiap kueri orde pertama, jawaban yang dikembalikan mendekati posterior sejati secara sembarangan dalam batas. Ada beberapa masalah rumit yang terlibat dalam merancang algoritma ini. Misalnya, algoritma MCMC tidak dapat mengambil sampel secara langsung dalam ruang kemungkinan dunia ketika ukuran dunia tersebut tidak terbatas; sebaliknya, ia mengambil sampel dunia terbatas dan parsial, dengan mengandalkan fakta bahwa hanya sejumlah objek terbatas yang dapat relevan dengan kueri dengan cara yang berbeda. Lebih jauh, transisi harus memungkinkan penggabungan dua objek menjadi satu atau pemisahan satu menjadi dua. (Rincian diberikan dalam referensi di akhir bab. Meskipun ada kerumitan ini, prinsip dasar yang ditetapkan dalam Persamaan (5.13) masih berlaku: probabilitas kalimat apa pun terdefinisi dengan baik dan dapat dihitung.

Penelitian di bidang ini masih dalam tahap awal, tetapi sudah menjadi jelas bahwa penalaran probabilistik tingkat pertama menghasilkan peningkatan yang luar biasa dalam efektivitas sistem AI dalam menangani informasi yang tidak pasti. Aplikasi potensial mencakup yang disebutkan di atas—visi komputer, pemahaman teks, dan analisis kecerdasan—serta banyak jenis interpretasi sensor lainnya.

5.7 PENDEKATAN LAIN UNTUK PENALARAN YANG TAK PASTI

Ilmu-ilmu lain (misalnya, fisika, genetika, dan ekonomi) telah lama mendukung probabilitas sebagai model ketidakpastian. Pada tahun 1819, Pierre Laplace berkata, "Teori probabilitas tidak lain hanyalah akal sehat yang direduksi menjadi kalkulasi." Pada tahun 1850, James Maxwell berkata, "Logika sejati untuk dunia ini adalah kalkulus Probabilitas, yang

memperhitungkan besarnya probabilitas yang ada, atau seharusnya ada, dalam pikiran orang yang berakal sehat."

Mengingat tradisi yang panjang ini, mungkin mengejutkan bahwa AI telah mempertimbangkan banyak alternatif untuk probabilitas. Sistem pakar paling awal tahun 1970-an mengabaikan ketidakpastian dan menggunakan penalaran logis yang ketat, tetapi segera menjadi jelas bahwa ini tidak praktis untuk sebagian besar domain dunia nyata. Generasi sistem pakar berikutnya (terutama dalam domain medis) menggunakan teknik probabilistik. Hasil awalnya menjanjikan, tetapi tidak ditingkatkan karena jumlah probabilitas eksponensial yang diperlukan dalam distribusi gabungan penuh. (Algoritma jaringan Bayesian yang efisien belum dikenal saat itu.) Akibatnya, pendekatan probabilistik tidak lagi disukai dari sekitar tahun 1975 hingga 1988, dan berbagai alternatif terhadap probabilitas dicoba karena berbagai alasan:

- Satu pandangan umum adalah bahwa teori probabilitas pada dasarnya numerik, sedangkan penalaran manusia yang menghakimi lebih "kualitatif." Tentu saja, kita tidak secara sadar menyadari melakukan kalkulasi numerik terhadap derajat keyakinan. (Kita juga tidak menyadari melakukan penyatuan, tetapi kita tampaknya mampu melakukan semacam penalaran logis.) Mungkin saja kita memiliki semacam derajat keyakinan numerik yang dikodekan secara langsung dalam kekuatan koneksi dan aktivasi dalam neuron kita. Dalam hal itu, kesulitan akses sadar terhadap kekuatan tersebut tidak mengejutkan. Perlu juga dicatat bahwa mekanisme penalaran kualitatif dapat dibangun langsung di atas teori probabilitas, sehingga argumen "tanpa angka" terhadap probabilitas tidak memiliki kekuatan yang berarti. Meskipun demikian, beberapa skema kualitatif memiliki daya tarik tersendiri. Salah satu yang paling banyak dipelajari adalah **penalaran baku**, yang memperlakukan kesimpulan bukan sebagai "dipercaya sampai tingkat tertentu," tetapi sebagai "dipercaya sampai ditemukan alasan yang lebih baik untuk mempercayai sesuatu yang lain."
- Pendekatan **berbasis aturan** terhadap ketidakpastian juga telah dicoba. Pendekatan semacam itu berharap untuk membangun keberhasilan sistem berbasis aturan yang logis, tetapi menambahkan semacam "faktor kesalahan" pada setiap aturan untuk mengakomodasi ketidakpastian. Metode ini dikembangkan pada pertengahan tahun 1970-an dan menjadi dasar bagi sejumlah besar sistem pakar dalam bidang kedokteran dan bidang lainnya.
- Satu bidang yang belum kita bahas sejauh ini adalah pertanyaan tentang **ketidaktahuan**, sebagai lawan dari ketidakpastian. Pertimbangkan melempar koin. Jika kita tahu bahwa koin itu adil, maka probabilitas 0,5 untuk sisi kepala adalah wajar. Jika kita tahu bahwa koin itu bias, tetapi kita tidak tahu ke arah mana, maka 0,5 untuk sisi kepala lagi-lagi wajar. Jelas, kedua kasus itu berbeda, tetapi probabilitas hasil tampaknya tidak membedakannya. **Teori Dempster-Shafer** menggunakan derajat **keyakinan bernilai interval** untuk merepresentasikan pengetahuan agen tentang probabilitas suatu proposisi.

- Probabilitas membuat komitmen ontologis yang sama seperti logika: bahwa proposisi itu benar atau salah di dunia, bahkan jika agen tidak yakin yang mana yang benar. Para peneliti dalam **logika fuzzy** telah mengusulkan ontologi yang memungkinkan **ketidakjelasan**: bahwa suatu proposisi dapat "semacam" benar. Ketidakjelasan dan ketidakpastian sebenarnya adalah masalah yang ortogonal.

Tiga subbagian berikutnya membahas beberapa pendekatan ini dengan sedikit lebih mendalam. Kami tidak akan memberikan materi teknis yang terperinci, tetapi kami mengutip referensi untuk studi lebih lanjut.

Metode berbasis aturan untuk penalaran yang tidak pasti

Sistem berbasis aturan muncul dari karya awal tentang sistem praktis dan intuitif untuk inferensi logis. Sistem logis secara umum, dan sistem berbasis aturan logis secara khusus, memiliki tiga sifat yang diinginkan:

- **Lokalitas**: Dalam sistem logis, setiap kali kita memiliki aturan dalam bentuk $A \Rightarrow B$, kita dapat menyimpulkan B , dengan bukti A , *tanpa mengkhawatirkan aturan lainnya*. Dalam sistem probabilistik, kita perlu mempertimbangkan *semua* bukti.
- **Detasemen**: Setelah bukti logis ditemukan untuk proposisi B , proposisi tersebut dapat digunakan terlepas dari bagaimana proposisi tersebut diturunkan. Artinya, proposisi tersebut dapat **dilepaskan** dari pembenarannya. Di sisi lain, dalam menangani probabilitas, sumber bukti untuk suatu keyakinan penting untuk penalaran selanjutnya.
- **Fungsionalitas kebenaran**: Dalam logika, kebenaran kalimat kompleks dapat dihitung dari kebenaran komponennya. Kombinasi probabilitas tidak bekerja dengan cara ini, kecuali di bawah asumsi independensi global yang kuat.

Telah ada beberapa upaya untuk merancang skema penalaran tak pasti yang mempertahankan keunggulan ini. Idenya adalah untuk mengaitkan derajat keyakinan pada proposisi dan aturan dan merancang skema lokal murni untuk menggabungkan dan menyebarkan derajat keyakinan tersebut. Skema tersebut juga berfungsi sebagai fungsi kebenaran; misalnya, derajat keyakinan pada $A \vee B$ adalah fungsi dari keyakinan pada A dan keyakinan pada B .

Kabar buruk untuk sistem berbasis aturan adalah bahwa sifat *lokalitas*, *keterpisahan*, dan *fungsi kebenaran sama sekali tidak sesuai untuk penalaran tak pasti*. Mari kita lihat fungsi kebenaran terlebih dahulu. Misalkan H_1 adalah peristiwa bahwa lemparan koin yang adil menghasilkan sisi kepala, misalkan T_1 adalah peristiwa bahwa koin menghasilkan sisi ekor pada lemparan yang sama, dan misalkan H_2 adalah peristiwa bahwa koin menghasilkan sisi kepala pada lemparan kedua. Jelas, ketiga peristiwa tersebut memiliki probabilitas yang sama, 0,5, sehingga sistem fungsi kebenaran harus menetapkan keyakinan yang sama pada disjungsi dari dua peristiwa tersebut. Namun kita dapat melihat bahwa probabilitas disjungsi bergantung pada peristiwa itu sendiri dan bukan hanya pada probabilitasnya:

$P(A)$	$P(B)$	$P(A \vee B)$
$P(H_1) = 0.5$	$P(H_1) = 0.5$ $P(T_1) = 0.5$	$P(H_1 \vee H_1) = 0.50$ $P(H_1 \vee T_1) = 1.00$

	$P(H_2) = 0.5$	$P(H_1 \vee H_2) = 0.75$

Keadaannya makin buruk saat kita merangkai bukti bersama-sama. Sistem fungsional kebenaran memiliki *aturan* dalam bentuk $A \rightarrow B$ yang memungkinkan kita menghitung keyakinan pada B sebagai fungsi dari keyakinan pada aturan dan keyakinan pada A . Sistem rantai maju dan mundur dapat dirancang. Keyakinan pada aturan diasumsikan konstan dan biasanya ditentukan oleh perekayasa pengetahuan—misalnya, sebagai $A \rightarrow_{0,9} B$.

Pertimbangkan situasi rumput basah dari Gambar 5.12(a). Jika kita ingin mampu melakukan penalaran kausal dan diagnostik, kita memerlukan kedua aturan tersebut

$$Rain \rightarrow WetGrass \quad \text{dan} \quad WetGrass \rightarrow Rain.$$

Kedua aturan ini membentuk lingkaran umpan balik: bukti untuk *Rain* meningkatkan kepercayaan pada *WetGrass*, yang pada gilirannya meningkatkan kepercayaan pada *Rain* lebih jauh lagi. Jelas, sistem penalaran yang tidak pasti perlu melacak jalur yang dilalui bukti untuk disebarkan.

Penalaran interkausal (atau menjelaskan) juga rumit. Pertimbangkan apa yang terjadi ketika kita memiliki dua aturan tersebut

$$Sprinkler \rightarrow WetGrass \quad \text{Dan} \quad WetGrass \rightarrow Rain.$$

Misalkan kita melihat alat penyiram menyala. Dengan merangkai aturan kita, hal ini meningkatkan keyakinan bahwa rumput akan basah, yang pada gilirannya meningkatkan keyakinan bahwa hujan. Namun, ini menggelikan: fakta bahwa alat penyiram menyala menjelaskan rumput basah dan seharusnya *mengurangi* keyakinan pada hujan. Sistem fungsional-kebenaran bertindak seolah-olah juga meyakini $Sprinkler \rightarrow Rain$.

Dengan kesulitan-kesulitan ini, bagaimana sistem fungsional-kebenaran dapat dibuat berguna dalam praktik?. Jawabannya terletak pada pembatasan tugas dan rekayasa basis aturan secara cermat sehingga interaksi yang tidak diinginkan tidak terjadi. Contoh paling terkenal dari sistem fungsional-kebenaran untuk penalaran yang tidak pasti adalah model **faktor kepastian**, yang dikembangkan untuk program diagnosis medis MYCIN dan digunakan secara luas dalam sistem pakar pada akhir tahun 1970-an dan 1980-an. Hampir semua penggunaan faktor kepastian melibatkan perangkat aturan yang murni diagnostik (seperti dalam MYCIN) atau murni kausal. Lebih jauh lagi, bukti dimasukkan hanya pada "akar" dari perangkat aturan, dan sebagian besar perangkat aturan terhubung secara tunggal. Heckerman (1986) telah menunjukkan bahwa, dalam keadaan ini, variasi kecil pada inferensi faktor kepastian sama persis dengan inferensi Bayesian pada polytrees. Dalam keadaan lain, faktor kepastian dapat menghasilkan derajat keyakinan yang sangat tidak tepat melalui penghitungan bukti yang berlebihan. Ketika perangkat aturan menjadi lebih besar, interaksi yang tidak diinginkan antara aturan menjadi lebih umum, dan praktisi menemukan bahwa faktor kepastian dari banyak aturan lain harus "diubah" ketika aturan baru ditambahkan. Karena

alasan ini, jaringan Bayesian sebagian besar telah menggantikan metode berbasis aturan untuk penalaran yang tidak pasti.

Mewakili ketidaktauan: Teori Dempster–Shafer

Teori Dempster–Shafer dirancang untuk menangani perbedaan antara **ketidakpastian** dan **ketidaktauan**. Daripada menghitung probabilitas suatu proposisi, teori ini menghitung probabilitas bahwa bukti mendukung proposisi tersebut. Ukuran keyakinan ini disebut **fungsi keyakinan**, ditulis $Bel(X)$.

Kita kembali ke lemparan koin untuk contoh fungsi keyakinan. Misalkan Anda mengambil koin dari saku pesulap. Mengingat koin itu mungkin adil atau mungkin tidak adil, keyakinan apa yang harus Anda kaitkan dengan peristiwa munculnya sisi kepala? Teori Dempster–Shafer mengatakan bahwa karena Anda tidak memiliki bukti apa pun, Anda harus mengatakan bahwa keyakinan $Bel(Heads) = 0$ dan juga bahwa $Bel(\neg Heads) = 0$. Hal ini membuat sistem penalaran Dempster–Shafer menjadi skeptis dengan cara yang memiliki daya tarik intuitif. Sekarang anggaplah Anda memiliki seorang ahli yang siap memberikan kesaksian dengan kepastian 90% bahwa koin tersebut adil (yaitu, ia 90% yakin bahwa $P(Heads) = 0,5$). Maka teori Dempster–Shafer memberikan $Bel(Heads) = 0,9 \times 0,5 = 0,45$ dan demikian pula $Bel(\neg Heads) = 0,45$. Masih ada "kesenjangan" 10 poin persentase yang tidak diperhitungkan oleh bukti.

Landasan matematika teori Dempster–Shafer memiliki cita rasa yang sama dengan teori probabilitas; perbedaan utamanya adalah, alih-alih menetapkan probabilitas pada kemungkinan dunia, teori tersebut menetapkan **massa** pada *himpunan* kemungkinan dunia, yaitu, pada kejadian. Massa tersebut tetap harus berjumlah 1 untuk semua kejadian yang mungkin. $Bel(A)$ didefinisikan sebagai jumlah massa untuk semua kejadian yang merupakan bagian dari (yaitu, yang memerlukan) A , termasuk A itu sendiri. Dengan definisi ini, $Bel(A)$ dan $Bel(\neg A)$ berjumlah *paling banyak* 1, dan celah—interval antara $Bel(A)$ dan $1 - Bel(\neg A)$ —sering ditafsirkan sebagai pembatas probabilitas A .

Seperti halnya penalaran baku, ada masalah dalam menghubungkan keyakinan dengan tindakan. Setiap kali ada celah dalam keyakinan, maka masalah keputusan dapat didefinisikan sedemikian rupa sehingga sistem Dempster–Shafer tidak dapat membuat keputusan. Faktanya, gagasan utilitas dalam model Dempster–Shafer belum dipahami dengan baik karena makna massa dan keyakinan itu sendiri belum dipahami. Pearl (1988) berpendapat bahwa $Bel(A)$ harus ditafsirkan bukan sebagai tingkat keyakinan pada A tetapi sebagai probabilitas yang ditetapkan untuk semua kemungkinan dunia (sekarang ditafsirkan sebagai teori logis) di mana A dapat *dibuktikan*. Meskipun ada kasus di mana kuantitas ini mungkin menarik, itu tidak sama dengan probabilitas bahwa A benar.

Analisis Bayesian dari contoh lemparan koin akan menunjukkan bahwa tidak diperlukan formalisme baru untuk menangani kasus-kasus seperti itu. Model akan memiliki dua variabel: *Bias* koin (angka antara 0 dan 1, di mana 0 adalah koin yang selalu menunjukkan sisi belakang dan 1 adalah koin yang selalu menunjukkan sisi depan) dan hasil dari *lemparan* berikutnya. Distribusi probabilitas sebelumnya untuk *Bias* akan mencerminkan keyakinan kita berdasarkan sumber koin (kantong pesulap): beberapa probabilitas kecil bahwa itu adil dan beberapa

probabilitas bahwa itu sangat bias terhadap sisi depan atau sisi belakang. Distribusi kondisional $P(\text{Flip} | \text{Bias})$ hanya mendefinisikan bagaimana bias beroperasi. Jika $P(\text{Bias})$ simetris sekitar 0,5, maka probabilitas sebelumnya kita untuk lemparan adalah

$$P(\text{Flip} = \text{heads}) = \int_0^1 P(\text{Bias} = x)P(\text{Flip} = \text{heads} | \text{Bias} = x) dx = 0.5.$$

Ini adalah prediksi yang sama seperti jika kita sangat yakin bahwa koin itu adil, tetapi itu tidak berarti bahwa teori probabilitas memperlakukan kedua situasi secara identik. Perbedaan muncul *setelah* lemparan koin dalam menghitung distribusi posterior untuk *Bias*. Jika koin itu berasal dari bank, maka melihatnya muncul sisi kepala tiga kali berturut-turut hampir tidak akan berpengaruh pada keyakinan kuat kita sebelumnya tentang kewajarannya; tetapi jika koin itu berasal dari kantong pesulap, bukti yang sama akan mengarah pada keyakinan posterior yang lebih kuat bahwa koin itu bias ke sisi kepala. Dengan demikian, pendekatan Bayesian mengekspresikan "ketidaktahuan" kita dalam hal bagaimana keyakinan kita akan berubah dalam menghadapi pengumpulan informasi di masa mendatang.

Mewakili ketidakjelasan: Himpunan fuzzy dan logika fuzzy

Teori himpunan fuzzy adalah cara untuk menentukan seberapa baik suatu objek memenuhi deskripsi yang tidak jelas. Misalnya, pertimbangkan proposisi "Nate tinggi." Apakah ini benar jika Nate tingginya 5' 10"? Kebanyakan orang akan ragu untuk menjawab "benar" atau "salah," lebih suka mengatakan, "agak." Perhatikan bahwa ini bukan masalah ketidakpastian tentang dunia luar—kita yakin akan tinggi Nate. Masalahnya adalah bahwa istilah linguistik "tinggi" tidak merujuk pada demarkasi tajam objek ke dalam dua kelas—ada derajat tinggi. Karena alasan ini, teori himpunan fuzzy sama sekali bukan metode untuk penalaran yang tidak pasti. Sebaliknya, teori himpunan fuzzy memperlakukan Tinggi sebagai predikat fuzzy dan mengatakan bahwa nilai kebenaran $Tall(Nate)$ adalah angka antara 0 dan 1, bukan hanya *True* atau *False*. Nama "himpunan fuzzy" berasal dari interpretasi predikat yang secara implisit mendefinisikan himpunan anggotanya—himpunan yang tidak memiliki batas yang tajam.

Logika fuzzy adalah metode untuk penalaran dengan ekspresi logis yang menggambarkan keanggotaan dalam himpunan fuzzy. Misalnya, kalimat kompleks $Tall(Nate) \wedge Heavy(Nate)$ memiliki nilai kebenaran fuzzy yang merupakan fungsi dari nilai kebenaran komponen-komponennya. Aturan standar untuk mengevaluasi kebenaran fuzzy, T , dari kalimat kompleks adalah

$$\begin{aligned} T(A \wedge B) &= \min(T(A), T(B)) \\ T(A \vee B) &= \max(T(A), T(B)) \\ T(\neg A) &= 1 - T(A). \end{aligned}$$

Logika fuzzy karenanya merupakan sistem fungsional kebenaran—fakta yang menyebabkan kesulitan serius. Misalnya, misalkan $T(Tall(Nate)) = 0,6$ dan $T(Heavy(Nate)) = 0,4$.

Maka kita memiliki $T(Tall(Nate) \wedge heavy(Nate)) = 0,4$, yang tampaknya masuk akal, tetapi kita juga memperoleh hasil $T(Tall(Nate) \wedge \neg Tall(Nate)) = 0,4$, yang tidak masuk akal. Jelas, masalah muncul dari ketidakmampuan pendekatan fungsional kebenaran untuk memperhitungkan korelasi atau antikorelasi di antara proposisi komponen. Kontrol fuzzy adalah metodologi untuk membangun sistem kontrol di mana pemetaan antara parameter input dan output bernilai riil direpresentasikan oleh aturan fuzzy.

Kontrol fuzzy telah sangat berhasil dalam produk komersial seperti transmisi otomatis, kamera video, dan alat cukur listrik. Kritik berpendapat bahwa aplikasi ini berhasil karena memiliki basis aturan yang kecil, tidak ada rangkaian inferensi, dan parameter yang dapat disetel yang dapat disesuaikan untuk meningkatkan kinerja sistem. Fakta bahwa aplikasi ini diimplementasikan dengan operator fuzzy mungkin merupakan faktor yang tidak terkait dengan keberhasilannya; kuncinya adalah menyediakan cara yang ringkas dan intuitif untuk menentukan fungsi bernilai riil yang diinterpolasi dengan lancar.

Ada upaya untuk memberikan penjelasan tentang logika fuzzy dalam hal teori probabilitas. Salah satu idenya adalah melihat pernyataan seperti "*Nate Tall*" sebagai pengamatan diskrit yang dibuat mengenai variabel tersembunyi yang berkelanjutan, *Height* aktual Nate. Model probabilitas menentukan $P(\text{Pengamat mengatakan Nate Tall} \mid \text{Height})$, mungkin menggunakan **distribusi probit** seperti yang dijelaskan pada halaman sebelumnya. Distribusi posterior atas tinggi Nate kemudian dapat dihitung dengan cara yang biasa, misalnya, jika model tersebut merupakan bagian dari jaringan Bayesian hibrida. Pendekatan seperti itu tentu saja tidak fungsional-kebenaran. Misalnya, distribusi bersyarat

$$P(\text{Observer says Nate is tall and heavy} \mid \text{Height, Weight})$$

memungkinkan interaksi antara tinggi dan berat dalam penyebab pengamatan. Jadi, seseorang yang tingginya delapan kaki dan beratnya 190 pon sangat tidak mungkin disebut "tinggi dan berat," meskipun "delapan kaki" dihitung sebagai "tinggi" dan "190 pon" dihitung sebagai "berat."

Predikat fuzzy juga dapat diberikan interpretasi probabilistik dalam bentuk **himpunan acak**—yaitu, variabel acak yang nilai-nilai yang mungkin adalah himpunan objek. Misalnya, *Tall* adalah himpunan acak yang nilai-nilai yang mungkin adalah himpunan orang. Probabilitas $P(Tall = S_1)$, di mana S_1 adalah himpunan orang tertentu, adalah probabilitas bahwa himpunan itu akan diidentifikasi sebagai "tinggi" oleh seorang pengamat. Kemudian probabilitas bahwa "Nate tinggi" adalah jumlah probabilitas semua himpunan yang Nate menjadi anggotanya.

Pendekatan jaringan Bayesian hibrida dan pendekatan himpunan acak tampaknya menangkap aspek-aspek ketidakjelasan tanpa memperkenalkan derajat kebenaran. Meskipun demikian, masih banyak isu terbuka mengenai representasi yang tepat dari pengamatan linguistik dan kuantitas kontinu—isu-isu yang telah diabaikan oleh sebagian besar orang di luar komunitas fuzzy.

5.8 RINGKASAN

Bab ini telah menjelaskan **jaringan Bayesian**, representasi yang dikembangkan dengan baik untuk pengetahuan yang tidak pasti. Jaringan Bayesian memainkan peran yang secara kasar mirip dengan logika proposisional untuk pengetahuan yang pasti.

- Jaringan Bayesian adalah grafik asiklik terarah yang simpulnya sesuai dengan variabel acak; setiap simpul memiliki distribusi kondisional untuk simpul tersebut, yang diberikan induknya.
- Jaringan Bayesian menyediakan cara ringkas untuk merepresentasikan hubungan independensi kondisional dalam domain.
- Jaringan Bayesian menentukan distribusi gabungan penuh; setiap entri gabungan didefinisikan sebagai produk dari entri yang sesuai dalam distribusi kondisional lokal. Jaringan Bayesian sering kali secara eksponensial lebih kecil daripada distribusi gabungan yang disebutkan secara eksplisit.
- Banyak distribusi kondisional dapat direpresentasikan secara kompak oleh keluarga distribusi kanonik. **Jaringan Bayesian hibrida**, yang mencakup variabel diskrit dan kontinu, menggunakan berbagai distribusi kanonik.
- Inferensi dalam jaringan Bayesian berarti menghitung distribusi probabilitas dari sekumpulan variabel kueri, dengan sekumpulan variabel bukti. Algoritme inferensi eksak, **seperti eliminasi variabel**, mengevaluasi jumlah produk probabilitas bersyarat seefisien mungkin.
- Dalam **polytree** (jaringan yang terhubung tunggal), inferensi eksak membutuhkan waktu linier dalam ukuran jaringan. Dalam kasus umum, masalahnya tidak dapat dipecahkan.
- Teknik aproksimasi stokastik seperti **pembobotan kemungkinan** dan **rantai Markov Monte Carlo** dapat memberikan estimasi yang wajar dari probabilitas posterior sebenarnya dalam jaringan dan dapat mengatasi jaringan yang jauh lebih besar daripada algoritme eksak.
- Teori probabilitas dapat dikombinasikan dengan ide representasional dari logika orde pertama untuk menghasilkan sistem yang sangat kuat untuk penalaran dalam ketidakpastian. **Model probabilitas relasional** (RPM) mencakup batasan representasional yang menjamin distribusi probabilitas yang terdefinisi dengan baik yang dapat dinyatakan sebagai jaringan Bayesian yang ekuivalen. **Model probabilitas alam semesta terbuka** menangani **ketidakpastian eksistensi** dan identitas, mendefinisikan distribusi probabilitas atas ruang tak terbatas dari dunia-dunia yang mungkin pada tingkat pertama.
- Berbagai sistem alternatif untuk penalaran dalam ketidakpastian telah disarankan. Secara umum, sistem **fungsional-kebenaran** tidak cocok untuk penalaran semacam itu.

BAB 6

PENALARAN PROBABILISTIK DARI WAKTU KE WAKTU

Agen dalam lingkungan yang dapat diamati sebagian harus mampu melacak keadaan saat ini, sejauh yang dimungkinkan oleh sensor mereka. Di Bagian ini kami menunjukkan metodologi untuk melakukannya: agen mempertahankan **keadaan keyakinan** yang mewakili keadaan dunia mana yang saat ini mungkin terjadi. Dari keadaan keyakinan dan **model transisi**, agen dapat memprediksi bagaimana dunia mungkin berevolusi pada langkah waktu berikutnya. Dari persepsi yang diamati dan **model sensor**, agen dapat memperbarui keadaan keyakinan. Ini adalah ide yang meluas: dalam Bab sebelumnya keadaan keyakinan diwakili oleh serangkaian keadaan yang disebutkan secara eksplisit. Pendekatan tersebut mendefinisikan keadaan keyakinan dalam hal keadaan dunia mana yang *mungkin terjadi*, tetapi tidak dapat mengatakan apa pun tentang keadaan mana yang *mungkin* atau *tidak mungkin* terjadi. Dalam bab ini, kami menggunakan teori probabilitas untuk mengukur tingkat keyakinan dalam elemen-elemen status keyakinan.

Seperti yang kami tunjukkan di Bagian 6.1,: dunia yang berubah dimodelkan menggunakan variabel untuk setiap aspek status dunia *di setiap titik waktu*. Model transisi dan sensor mungkin tidak pasti: model transisi menggambarkan distribusi probabilitas variabel pada waktu t , mengingat status dunia di masa lalu, sementara model sensor menggambarkan probabilitas setiap persepsi pada waktu t , mengingat status dunia saat ini. Bagian 6.2 mendefinisikan tugas-tugas inferensi dasar dan menggambarkan struktur umum algoritma inferensi untuk model temporal. Kemudian kami menggambarkan tiga jenis model tertentu: **model Markov tersembunyi**, **filter Kalman**, dan **jaringan Bayesian dinamis** (yang mencakup model Markov tersembunyi dan filter Kalman sebagai kasus khusus). Terakhir, Bagian 6.6 mengkaji masalah-masalah yang dihadapi saat melacak lebih dari satu hal.

6.1 WAKTU DAN KETIDAKPASTIAN

Kami telah mengembangkan teknik untuk penalaran probabilistik dalam konteks dunia *statis*, di mana setiap variabel acak memiliki satu nilai tetap. Misalnya, saat memperbaiki mobil, kami berasumsi bahwa apa pun yang rusak akan tetap rusak selama proses diagnosis; tugas kami adalah menyimpulkan kondisi mobil dari bukti yang diamati, yang juga tetap tidak berubah.

Sekarang pertimbangkan masalah yang sedikit berbeda: merawat pasien diabetes. Seperti dalam kasus perbaikan mobil, kita memiliki bukti seperti dosis insulin terkini, asupan makanan, pengukuran gula darah, dan tanda-tanda fisik lainnya. Tugasnya adalah menilai kondisi pasien saat ini, termasuk kadar gula darah dan kadar insulin aktual. Berdasarkan informasi ini, kita dapat membuat keputusan tentang asupan makanan dan dosis insulin pasien. Tidak seperti kasus perbaikan mobil, di sini aspek *dinamis* dari masalah tersebut sangat penting. Kadar gula darah dan pengukurannya dapat berubah dengan cepat seiring waktu,

tergantung pada asupan makanan terkini dan dosis insulin, aktivitas metabolisme, waktu, dan sebagainya. Untuk menilai kondisi terkini dari riwayat bukti dan untuk memprediksi hasil tindakan pengobatan, kita harus memodelkan perubahan ini.

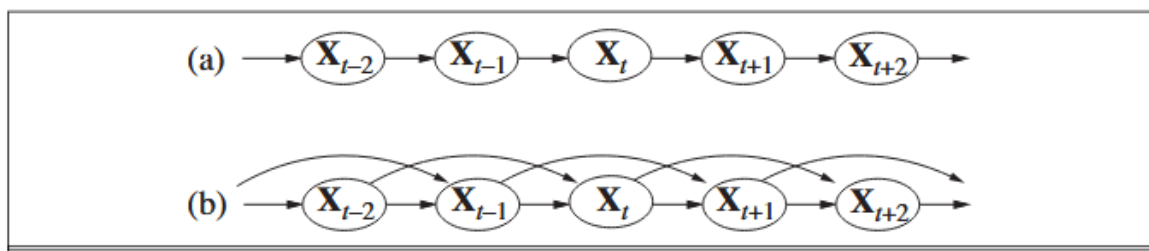
Pertimbangan yang sama muncul dalam banyak konteks lain, seperti melacak lokasi robot, melacak aktivitas ekonomi suatu negara, dan memahami urutan kata-kata yang diucapkan atau ditulis. Bagaimana situasi dinamis seperti ini dapat dimodelkan?

Kondisi dan pengamatan

Kita melihat dunia sebagai serangkaian cuplikan, atau **irisan waktu**, yang masing-masing berisi serangkaian variabel acak, beberapa dapat diamati dan beberapa tidak. Untuk menyederhanakannya, kita akan berasumsi bahwa himpunan bagian variabel yang sama dapat diamati di setiap irisan waktu (meskipun ini tidak sepenuhnya diperlukan dalam hal apa pun yang mengikutinya). Kita akan menggunakan $\mathbf{X}_t$ untuk menunjukkan himpunan variabel status pada waktu t , yang diasumsikan tidak dapat diamati, dan $\mathbf{E}_t$ untuk menunjukkan himpunan variabel bukti yang dapat diamati. Pengamatan pada waktu t adalah $\mathbf{E}_t = \mathbf{e}_t$ untuk beberapa himpunan nilai $\mathbf{e}_t$.

Pertimbangkan contoh berikut: Anda adalah penjaga keamanan yang ditempatkan di instalasi bawah tanah rahasia. Anda ingin tahu apakah hari ini hujan, tetapi satu-satunya akses Anda ke dunia luar terjadi setiap pagi ketika Anda melihat direktur datang dengan atau tanpa payung. Untuk setiap hari t , himpunan $\mathbf{E}_t$ dengan demikian berisi satu variabel bukti $Umbrella_t$ atau disingkat U_t (apakah payung muncul), dan himpunan $\mathbf{X}_t$ berisi satu variabel status $Rain_t$ atau disingkat R_t (apakah sedang hujan). Masalah lain dapat melibatkan kumpulan variabel yang lebih besar. Dalam contoh diabetes, kita mungkin memiliki variabel bukti, seperti $MeasuredBloodSugar_t$ dan $PulseRate_t$, dan variabel status, seperti $BloodSugar_t$ dan $StomachContents_t$. (Perhatikan bahwa $BloodSugar_t$ dan $MeasuredBloodSugar_t$ bukanlah variabel yang sama; ini adalah cara kita menangani pengukuran yang tidak jelas dari kuantitas aktual.)

Interval antara irisan waktu juga bergantung pada masalahnya. Untuk pemantauan diabetes, interval yang sesuai mungkin satu jam, bukan satu hari. Dalam bab ini, kita mengasumsikan interval antara irisan adalah tetap, sehingga kita dapat memberi label waktu dengan bilangan bulat. Kita akan mengasumsikan bahwa urutan status dimulai pada $t = 0$; karena berbagai alasan yang tidak menarik, kita akan mengasumsikan bahwa bukti mulai tiba pada $t = 1$, bukan $t = 0$. Oleh karena itu, dunia payung kita direpresentasikan oleh variabel status $R_0, R_1, R_2, \dots$ dan variabel bukti $U_1, U_2, \dots$. Kita akan menggunakan notasi $a:b$ untuk menunjukkan deretan bilangan bulat dari a hingga b (inklusif), dan notasi $\mathbf{X}_{a:b}$ untuk menunjukkan himpunan variabel dari $\mathbf{X}_a$ hingga $\mathbf{X}_b$. Misalnya, $U_{1:3}$ sesuai dengan variabel U_1, U_2, U_3 .



Gambar 6.1 (a) Struktur jaringan Bayesian yang sesuai dengan proses Markov orde pertama dengan keadaan yang didefinisikan oleh variabel X_t . (b) Proses Markov orde kedua.

Model transisi dan sensor

Setelah serangkaian variabel status dan bukti untuk masalah tertentu diputuskan, langkah berikutnya adalah menentukan bagaimana dunia berevolusi (model transisi) dan bagaimana variabel bukti mendapatkan nilainya (model sensor).

Model transisi menentukan distribusi probabilitas atas variabel status terbaru, dengan nilai sebelumnya, yaitu, $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1})$. Sekarang kita menghadapi masalah: serangkaian $\mathbf{X}_{0:t-1}$ tidak terbatas ukurannya saat t meningkat. Kita memecahkan masalah tersebut dengan membuat **asumsi Markov**—bahwa status saat ini hanya bergantung pada *sejumlah status sebelumnya yang terbatas dan tetap*. Proses yang memenuhi asumsi ini pertama kali dipelajari secara mendalam oleh ahli statistik Rusia Andrei Markov (1856–1922) dan disebut **proses Markov** atau **rantai Markov**. Proses ini hadir dalam berbagai bentuk; yang paling sederhana adalah proses **Markov order pertama**, di mana status saat ini hanya bergantung pada status sebelumnya dan tidak pada status sebelumnya. Dengan kata lain, sebuah negara menyediakan informasi yang cukup untuk membuat masa depan secara kondisional independen dari masa lalu, dan kita memiliki

$$\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1}). \quad (6.1)$$

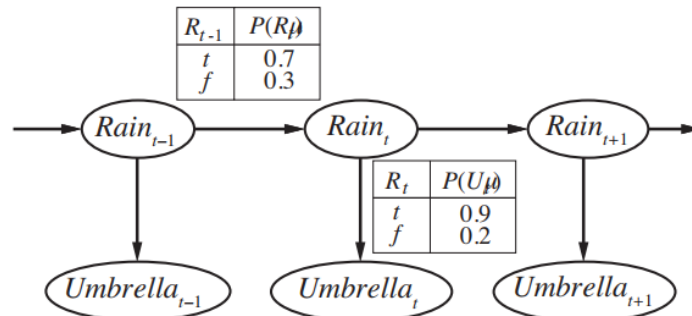
Oleh karena itu, dalam proses Markov orde pertama, model transisi adalah distribusi kondisional $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$. Model transisi untuk proses Markov orde kedua adalah distribusi kondisional $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$. Gambar 6.1 menunjukkan struktur jaringan Bayesian yang sesuai dengan proses Markov orde pertama dan kedua.

Bahkan dengan asumsi Markov masih ada masalah: ada banyak kemungkinan nilai t yang tak terhingga. Apakah kita perlu menentukan distribusi yang berbeda untuk setiap langkah waktu? Kita menghindari masalah ini dengan mengasumsikan bahwa perubahan dalam keadaan dunia disebabkan oleh **proses stasioner**—yaitu, proses perubahan yang diatur oleh hukum yang tidak berubah seiring waktu. (Jangan bingung antara *stasioner* dengan *statis*: dalam proses *statis*, keadaan itu sendiri tidak berubah.) Jadi, dalam dunia payung, probabilitas kondisional hujan, $\mathbf{P}(R_t | R_{t-1})$, sama untuk semua t , dan kita hanya perlu menentukan satu tabel probabilitas kondisional.

Sekarang untuk model sensor. Variabel bukti E_t dapat bergantung pada variabel sebelumnya serta variabel status saat ini, tetapi status apa pun yang layak harus cukup untuk menghasilkan nilai sensor saat ini. Jadi, kami membuat **asumsi Markov sensor** sebagai berikut:

$$P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t). \quad (15.2)$$

Jadi, $P(E_t | X_t)$ adalah model sensor kita (kadang-kadang disebut **model observasi**). Gambar 6.2 menunjukkan model transisi dan model sensor untuk contoh payung. Perhatikan arah ketergantungan antara status dan sensor: anak panah bergerak dari status dunia yang sebenarnya ke nilai sensor karena status dunia *menyebabkan* sensor mengambil nilai tertentu: hujan *menyebabkan* payung muncul. (Proses inferensi, tentu saja, bergerak ke arah lain; perbedaan antara arah ketergantungan yang dimodelkan dan arah inferensi adalah salah satu keuntungan utama jaringan Bayesian.)



Gambar 6.2 Struktur jaringan Bayesian dan distribusi kondisional yang menggambarkan dunia payung. Model transisi adalah $P(Rain_t | Rain_{t-1})$ dan model sensor adalah $P(Umbrella_t | Rain_t)$.

Selain menentukan model transisi dan sensor, kita perlu mengatakan bagaimana semuanya dimulai—distribusi probabilitas sebelumnya pada waktu 0, $P(X_0)$. Dengan itu, kita memiliki spesifikasi distribusi gabungan lengkap atas semua variabel, menggunakan Persamaan (6.2). Untuk t apa pun,

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^t P(X_i | X_{i-1}) P(E_i | X_i). \quad (6.3)$$

Tiga suku di sisi kanan adalah model status awal $P(X_0)$, model transisi $P(X_i | X_{i-1})$, dan model sensor $P(E_i | X_i)$.

Struktur pada Gambar 6.2 adalah proses Markov orde pertama—probabilitas hujan diasumsikan hanya bergantung pada apakah hujan turun pada hari sebelumnya. Apakah asumsi tersebut masuk akal bergantung pada domain itu sendiri. Asumsi Markov orde pertama menyatakan bahwa variabel status berisi *semua* informasi yang diperlukan untuk mengkarakterisasi distribusi probabilitas untuk irisan waktu berikutnya. Terkadang asumsi

tersebut benar-benar tepat—misalnya, jika sebuah partikel melakukan jalan acak sepanjang sumbu x , mengubah posisinya sebesar ± 1 pada setiap langkah waktu, maka menggunakan koordinat x sebagai status menghasilkan proses Markov orde pertama. Terkadang asumsi tersebut hanya perkiraan, seperti dalam kasus memprediksi hujan hanya berdasarkan apakah hujan turun pada hari sebelumnya. Ada dua cara untuk meningkatkan akurasi perkiraan:

1. Meningkatkan orde model proses Markov. Misalnya, kita dapat membuat model orde kedua dengan menambahkan $Rain_{t-2}$ sebagai induk dari $Rain_t$, yang mungkin memberikan prediksi yang sedikit lebih akurat. Misalnya, di Palo Alto, California, sangat jarang hujan turun lebih dari dua hari berturut-turut.
2. Menambah set variabel status. Misalnya, kita dapat menambahkan $Season_t$ untuk memungkinkan kita memasukkan catatan historis musim hujan, atau kita dapat menambahkan $Temperature_t$, $Humidity_t$ dan $Pressure_t$ (mungkin di berbagai lokasi) untuk memungkinkan kita menggunakan model fisik kondisi hujan.

Perhatikan bahwa menambahkan variabel status dapat meningkatkan daya prediksi sistem tetapi juga meningkatkan *persyaratan* prediksi: kita sekarang harus memprediksi variabel baru juga. Jadi, kita mencari himpunan variabel yang "mandiri", yang berarti kita harus memahami "fisika" proses yang dimodelkan. Persyaratan untuk pemodelan proses yang akurat jelas berkurang jika kita dapat menambahkan sensor baru (misalnya, pengukuran suhu dan tekanan) yang memberikan informasi langsung tentang variabel status baru.

Pertimbangkan, misalnya, masalah pelacakan robot yang berkeliaran secara acak di bidang $X - Y$. Seseorang mungkin mengusulkan bahwa posisi dan kecepatan adalah himpunan variabel status yang memadai: seseorang dapat menggunakan hukum Newton untuk menghitung posisi baru, dan kecepatan dapat berubah secara tidak terduga. Namun, jika robot bertenaga baterai, maka habisnya daya baterai cenderung memiliki efek sistematis pada perubahan kecepatan. Karena hal ini bergantung pada seberapa banyak daya yang digunakan oleh semua manuver sebelumnya, maka sifat Markov dilanggar. Kita dapat memulihkan sifat Markov dengan memasukkan level pengisian daya $Battery_t$ sebagai salah satu variabel status yang membentuk $\mathbf{X}_t$. Hal ini membantu dalam memprediksi gerakan robot, tetapi pada gilirannya memerlukan model untuk memprediksi $Battery_t$ dari $Battery_{t-1}$ dan kecepatan. Dalam beberapa kasus, hal itu dapat dilakukan dengan andal, tetapi lebih sering kita menemukan bahwa kesalahan terakumulasi seiring waktu. Dalam kasus tersebut, akurasi dapat ditingkatkan dengan *menambahkan sensor baru* untuk level baterai.

6.2 INFERENSI DALAM MODEL TEMPORAL

Setelah menyiapkan struktur model temporal generik, kita dapat merumuskan tugas-tugas inferensi dasar yang harus diselesaikan:

- **Penyaringan:** Ini adalah tugas menghitung **status keyakinan**—distribusi posterior atas status terkini—dengan semua bukti hingga saat ini. Penyaringan juga disebut **estimasi status**. Dalam contoh kita, kita ingin menghitung $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$. Dalam contoh payung, ini berarti menghitung probabilitas hujan hari ini, dengan semua pengamatan pembawa payung yang dilakukan sejauh ini. Penyaringan adalah apa yang dilakukan agen rasional

untuk melacak status terkini sehingga keputusan rasional dapat dibuat. Ternyata perhitungan yang hampir identik memberikan kemungkinan urutan bukti, $P(\mathbf{e}_{1:t})$.

- **Prediksi:** Ini adalah tugas menghitung distribusi posterior atas status *mendatang*, dengan semua bukti hingga saat ini. Artinya, kita ingin menghitung $P(X_{t+k} | \mathbf{e}_{1:t})$ untuk beberapa $k > 0$. Dalam contoh umum, ini mungkin berarti menghitung probabilitas hujan tiga hari dari sekarang, dengan mempertimbangkan semua pengamatan hingga saat ini. Prediksi berguna untuk mengevaluasi kemungkinan tindakan berdasarkan hasil yang diharapkan.
- **Smoothing:** Ini adalah tugas menghitung distribusi posterior atas status *lampau*, dengan mempertimbangkan semua bukti hingga saat ini. Yaitu, kita ingin menghitung $P(X_k | \mathbf{e}_{1:t})$ untuk beberapa k sehingga $0 \leq k < t$. Dalam contoh payung, ini mungkin berarti menghitung probabilitas hujan turun Rabu lalu, dengan mempertimbangkan semua pengamatan pembawa payung yang dilakukan hingga hari ini. Smoothing memberikan estimasi status yang lebih baik daripada yang tersedia pada saat itu, karena menyertakan lebih banyak bukti.
- **Penjelasan yang paling mungkin:** Dengan mempertimbangkan serangkaian pengamatan, kita mungkin ingin menemukan serangkaian status yang paling mungkin menghasilkan pengamatan tersebut. Yaitu, kita ingin menghitung $\operatorname{argmax}_{x_{1:t}} P(\mathbf{x}_{1:t} | \mathbf{e}_{1:t})$. Misalnya, jika payung muncul pada masing-masing dari tiga hari pertama dan tidak muncul pada hari keempat, maka penjelasan yang paling mungkin adalah hujan turun pada tiga hari pertama dan tidak turun pada hari keempat. Algoritme untuk tugas ini berguna dalam banyak aplikasi, termasuk pengenalan ucapan—yang tujuannya adalah menemukan urutan kata yang paling mungkin, berdasarkan serangkaian bunyi—dan rekonstruksi rangkaian bit yang dikirimkan melalui saluran yang bising.

Selain tugas inferensi ini, kami juga memiliki

- **Pembelajaran:** Model transisi dan sensor, jika belum diketahui, dapat dipelajari dari pengamatan. Sama seperti jaringan Bayesian statis, pembelajaran jaringan Bayes dinamis dapat dilakukan sebagai produk sampingan dari inferensi. Inferensi memberikan estimasi tentang transisi apa yang sebenarnya terjadi dan status apa yang menghasilkan pembacaan sensor, dan estimasi ini dapat digunakan untuk memperbarui model. Model yang diperbarui memberikan estimasi baru, dan proses berulang hingga konvergensi. Proses keseluruhan adalah contoh dari ekspektasi-maksimisasi atau **algoritma EM**.

Perhatikan bahwa pembelajaran memerlukan penghalusan, bukan penyaringan, karena penghalusan memberikan estimasi yang lebih baik tentang status proses. Pembelajaran dengan penyaringan dapat gagal untuk konvergen dengan benar; pertimbangkan, misalnya, masalah belajar memecahkan kasus pembunuhan: kecuali Anda seorang saksi mata, penghalusan *selalu* diperlukan untuk menyimpulkan apa yang terjadi di tempat kejadian pembunuhan dari variabel yang dapat diamati.

Bagian selanjutnya dari bagian ini menjelaskan algoritme generik untuk empat tugas inferensi, terlepas dari jenis model tertentu yang digunakan. Perbaikan khusus untuk setiap model dijelaskan di bagian berikutnya.

Penyaringan dan prediksi

Algoritma penyaringan yang berguna perlu mempertahankan estimasi status terkini dan memperbaruinya, daripada menelusuri kembali seluruh riwayat persepsi untuk setiap pembaruan. (Jika tidak, biaya setiap pembaruan meningkat seiring berjalannya waktu.) Dengan kata lain, mengingat hasil penyaringan hingga waktu t , agen perlu menghitung hasil untuk $t + 1$ dari bukti baru $\mathbf{e}_{t+1}$,

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})),$$

untuk beberapa fungsi f . Proses ini disebut **estimasi rekursif**. Kita dapat melihat perhitungan yang terdiri dari dua bagian: pertama, distribusi status saat ini diproyeksikan maju dari t ke $t + 1$; kemudian diperbarui menggunakan bukti baru $\mathbf{e}_{t+1}$. Proses dua bagian ini muncul dengan cukup sederhana ketika rumus disusun ulang:

$$\begin{aligned} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) && \text{(Membagi bukti)} \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) && \text{(Menggunakan kaidah Bayes)} \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) && \text{(berdasarkan asumsi Markov sensor)} \end{aligned} \quad (6.4)$$

Di sini dan di seluruh bab ini, α adalah konstanta normalisasi yang digunakan untuk membuat probabilitas berjumlah 1. Istilah kedua, $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ mewakili prediksi satu langkah dari status berikutnya, dan istilah pertama memperbaruinya dengan bukti baru; perhatikan bahwa $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ dapat diperoleh langsung dari model sensor. Sekarang kita memperoleh prediksi satu langkah untuk status berikutnya dengan mengkondisikan status saat ini $\mathbf{X}_t$:

$$\begin{aligned} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t}) && \text{(Asumsi Markov).} \end{aligned} \quad (6.5)$$

Dalam penjumlahan, faktor pertama berasal dari model transisi dan faktor kedua berasal dari distribusi status terkini. Oleh karena itu, kita memiliki formulasi rekursif yang diinginkan. Kita dapat menganggap estimasi tersaring $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ sebagai “pesan” $\mathbf{f}_{1:t}$ yang disebarkan ke depan sepanjang urutan, dimodifikasi oleh setiap transisi dan diperbarui oleh setiap observasi baru. Prosesnya diberikan oleh

$$\mathbf{f}_{1:t+1} = \alpha \text{Forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1}),$$

di mana FORWARD mengimplementasikan pembaruan yang dijelaskan dalam Persamaan (6.5) dan proses dimulai dengan $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0)$. Ketika semua variabel status bersifat diskrit, waktu untuk setiap pembaruan bersifat konstan (yaitu, tidak bergantung pada t), dan ruang yang dibutuhkan juga konstan. (Tentu saja, konstanta bergantung pada ukuran ruang status dan jenis model temporal tertentu yang dimaksud.) *Persyaratan waktu dan ruang untuk pembaruan harus konstan jika agen dengan memori terbatas harus melacak distribusi status terkini melalui urutan pengamatan yang tidak terbatas.*

Mari kita ilustrasikan proses penyaringan untuk dua langkah dalam contoh dasar (Gambar 6.2.) Yaitu, kita akan menghitung $\mathbf{P}(R_2|u_{1:2})$ sebagai berikut:

- Pada hari ke-0, kita tidak memiliki pengamatan, hanya keyakinan sebelumnya dari petugas keamanan; mari kita asumsikan bahwa terdiri dari $\mathbf{P}(R_0) = \langle 0,5,0,5 \rangle$.
- Pada hari ke-1, payung muncul, jadi $U_1 = \text{True}$. Prediksi dari $t = 0$ hingga $t = 1$ adalah

$$\begin{aligned}\mathbf{P}(R_1) &= \sum_{r_0} \mathbf{P}(R_1 | r_0) \mathbf{P}(r_0) \\ &= \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle.\end{aligned}$$

Kemudian langkah pembaruan hanya dikalikan dengan probabilitas bukti untuk $t = 1$ dan dinormalisasi, seperti yang ditunjukkan pada Persamaan (6.4):

$$\begin{aligned}\mathbf{P}(R_1 | u_1) &= \alpha \mathbf{P}(u_1 | R_1) \mathbf{P}(R_1) = \alpha \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle \\ &= \alpha \langle 0.45, 0.1 \rangle \approx \langle 0.818, 0.182 \rangle.\end{aligned}$$

- Pada hari ke-2, payung muncul, jadi $U_2 = \text{True}$. Prediksi dari $t = 1$ hingga $t = 2$ adalah

$$\begin{aligned}\mathbf{P}(R_2 | u_1) &= \sum_{r_1} \mathbf{P}(R_2 | r_1) \mathbf{P}(r_1 | u_1) \\ &= \langle 0.7, 0.3 \rangle \times 0.818 + \langle 0.3, 0.7 \rangle \times 0.182 \approx \langle 0.627, 0.373 \rangle,\end{aligned}$$

dan memperbaruinya dengan bukti untuk $t = 2$ memberikan

$$\begin{aligned}\mathbf{P}(R_2 | u_1, u_2) &= \alpha \mathbf{P}(u_2 | R_2) \mathbf{P}(R_2 | u_1) = \alpha \langle 0.9, 0.2 \rangle \langle 0.627, 0.373 \rangle \\ &= \alpha \langle 0.565, 0.075 \rangle \approx \langle 0.883, 0.117 \rangle.\end{aligned}$$

Secara intuitif, kemungkinan hujan meningkat dari hari ke-1 hingga ke-2 karena hujan terus-menerus turun. Tugas **prediksi** dapat dilihat hanya sebagai penyaringan tanpa penambahan bukti baru. Faktanya, proses penyaringan sudah mencakup prediksi satu langkah, dan mudah untuk memperoleh perhitungan rekursif berikut untuk memprediksi keadaan pada $t + k + 1$ dari prediksi untuk $t + k$:

$$P(\mathbf{X}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} P(\mathbf{X}_{t+k+1} | \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t}) \quad (6.6)$$

Tentu saja, perhitungan ini hanya melibatkan model transisi dan bukan model sensor.

Menarik untuk mempertimbangkan apa yang terjadi saat kita mencoba memprediksi lebih jauh dan lebih jauh ke masa depan. Seperti yang ditunjukkan, distribusi hujan yang diprediksi konvergen ke titik tetap $\langle 0,5, 0,5 \rangle$, setelah itu tetap konstan sepanjang waktu. Ini adalah **distribusi stasioner** dari proses Markov yang didefinisikan oleh model transisi. Banyak yang diketahui tentang sifat-sifat distribusi tersebut dan tentang **waktu pencampuran**—kira-kira, waktu yang dibutuhkan untuk mencapai titik tetap. Dalam praktiknya, ini akan gagal dalam setiap upaya untuk memprediksi *keadaan sebenarnya* untuk sejumlah langkah yang lebih dari sebagian kecil waktu pencampuran, kecuali jika distribusi stasioner itu sendiri sangat memuncak di area kecil ruang keadaan. Semakin banyak ketidakpastian dalam model transisi, semakin pendek waktu pencampuran dan semakin kabur masa depan.

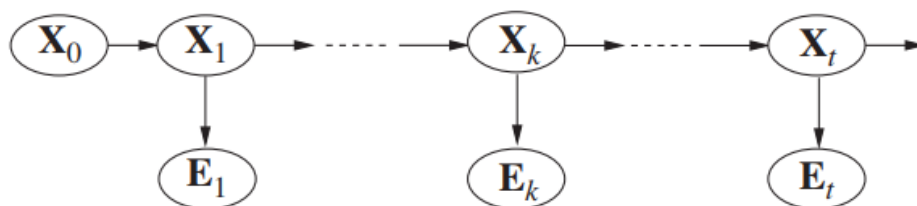
Selain penyaringan dan prediksi, kita dapat menggunakan rekursi maju untuk menghitung **kemungkinan** urutan bukti, $P(\mathbf{e}_{1:t})$. Ini adalah kuantitas yang berguna jika kita ingin membandingkan berbagai model temporal yang mungkin telah menghasilkan urutan bukti yang sama (misalnya, dua model berbeda untuk persistensi hujan). Untuk rekursi ini, kita menggunakan pesan kemungkinan $\ell_{1:t}(\mathbf{X}_t) = P(\mathbf{X}_t, \mathbf{e}_{1:t})$. Ini adalah latihan sederhana untuk menunjukkan bahwa kalkulasi pesan identik dengan kalkulasi untuk penyaringan:

$$\ell_{1:t+1} = \text{FORWARD}(\ell_{1:t}, \mathbf{e}_{t+1}).$$

Setelah menghitung $\ell_{1:t}$, kita memperoleh kemungkinan sebenarnya dengan menjumlahkan $\mathbf{X}_t$:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \ell_{1:t}(\mathbf{x}_t). \quad (6.7)$$

Perhatikan bahwa pesan kemungkinan mewakili probabilitas urutan bukti yang semakin panjang seiring berjalannya waktu dan karenanya menjadi semakin kecil secara numerik, yang mengarah ke masalah underflow dengan aritmatika floating-point. Ini adalah masalah penting dalam praktik, tetapi kita tidak akan membahas solusinya di sini.



Gambar 6.3 Perataan menghitung $P(\mathbf{X}_k | \mathbf{e}_{1:t})$, distribusi posterior keadaan pada suatu waktu lampau k yang diberikan urutan lengkap pengamatan dari 1 hingga t .

Smoothing

Seperti yang telah kami katakan sebelumnya, smoothing adalah proses menghitung distribusi atas keadaan lampau yang diberikan bukti hingga saat ini; yaitu, $P(\mathbf{X}_k | \mathbf{e}_{1:t})$ untuk $0 \leq k < t$. Untuk mengantisipasi pendekatan penyampaian pesan rekursif lainnya, kita dapat membagi perhitungan menjadi dua bagian—bukti hingga k dan bukti dari $k + 1$ hingga t ,

$$\begin{aligned}
 P(\mathbf{X}_k | \mathbf{e}_{1:t}) &= P(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
 &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) && \text{(menggunakan kaidah Bayes)} \\
 &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) && \text{(menggunakan independensi bersyarat)} \\
 &= \alpha \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t}.
 \end{aligned} \tag{6.8}$$

di mana “ $\times$ ” merupakan perkalian titik demi titik vektor. Di sini kita telah mendefinisikan pesan “mundur” $\mathbf{b}_{k+1:t} = P(\mathbf{e}_{k+1:t} | \mathbf{X}_k)$, yang analog dengan pesan maju $\mathbf{f}_{1:k}$. Pesan maju $\mathbf{f}_{1:k}$ dapat dihitung dengan memfilter maju dari 1 hingga k , seperti yang diberikan oleh Persamaan (6.5). Ternyata pesan mundur $\mathbf{b}_{k+1:t}$ dapat dihitung dengan proses rekursif yang *berjalan mundur* dari t :

$$\begin{aligned}
 P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k) && \text{(Pengondisian pada } \mathbf{X}_{k+1}) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) && \text{(berdasarkan independensi bersyarat)} \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k)
 \end{aligned} \tag{6.9}$$

di mana langkah terakhir diikuti oleh independensi bersyarat $\mathbf{e}_{k+1}$ dan $\mathbf{e}_{k+2:t}$, diberikan $\mathbf{X}_{k+1}$. Dari tiga faktor dalam penjumlahan ini, yang pertama dan ketiga diperoleh langsung dari model, dan yang kedua adalah “panggilan rekursif.” Dengan menggunakan notasi pesan, kita memiliki

$$\mathbf{b}_{k+1:t} = \text{BACKWARD}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1}),$$

di mana BACKWARD mengimplementasikan pembaruan yang dijelaskan dalam Persamaan (6.9). Seperti halnya rekursi maju, waktu dan ruang yang dibutuhkan untuk setiap pembaruan adalah konstan dan dengan demikian tidak bergantung pada t .

Kita sekarang dapat melihat bahwa kedua suku dalam Persamaan (6.8) dapat dihitung dengan rekursi melalui waktu, satu berjalan maju dari 1 hingga k dan menggunakan persamaan penyaringan (15.5) dan yang lainnya berjalan mundur dari t hingga $k + 1$ dan

menggunakan Persamaan (6.9). Perhatikan bahwa fase mundur diinisialisasi dengan $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t} | \mathbf{X}_t) = \mathbf{P}(\cdot | \mathbf{X}_t) \mathbf{1}$, di mana $\mathbf{1}$ adalah vektor 1s. (Karena $\mathbf{e}_{t+1:t}$ adalah deret kosong, maka peluang untuk mengamatinnya adalah 1.)

Sekarang mari kita terapkan algoritma ini pada contoh payung, dengan menghitung estimasi yang dihaluskan untuk peluang hujan pada waktu $k = 1$, dengan mempertimbangkan pengamatan payung pada hari ke-1 dan ke-2. Dari Persamaan (6.8), hal ini diberikan oleh

$$\mathbf{P}(R_1 | u_1, u_2) = \alpha \mathbf{P}(R_1 | u_1) \mathbf{P}(u_2 | R_1). \quad (6.10)$$

Istilah pertama yang sudah kita ketahui adalah $\langle 0.818, 0.182 \rangle$, dari proses penyaringan maju yang dijelaskan sebelumnya. Istilah kedua dapat dihitung dengan menerapkan rekursi mundur dalam Persamaan (6.9):

$$\begin{aligned} \mathbf{P}(u_2 | R_1) &= \sum_{r_2} P(u_2 | r_2) P(\cdot | r_2) \mathbf{P}(r_2 | R_1) \\ &= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle) = \langle 0.69, 0.41 \rangle. \end{aligned}$$

Dengan memasukkan hal ini ke Persamaan (6.10), kita menemukan bahwa estimasi yang dihaluskan untuk hujan pada hari ke-1 adalah

$$\mathbf{P}(R_1 | u_1, u_2) = \alpha \langle 0.818, 0.182 \rangle \times \langle 0.69, 0.41 \rangle \approx \langle 0.883, 0.117 \rangle.$$

Dengan demikian, estimasi yang dihaluskan untuk hujan pada hari ke-1 *lebih tinggi* daripada estimasi yang difilter (0,818) dalam kasus ini. Hal ini karena payung pada hari ke-2 membuat kemungkinan hujan pada hari ke-2 lebih besar; sebaliknya, karena hujan cenderung bertahan, maka kemungkinan hujan pada hari ke-1 lebih besar.

Baik rekursi maju maupun mundur membutuhkan waktu yang konstan per langkah; oleh karena itu, kompleksitas waktu penghalusan berkenaan dengan bukti $\mathbf{e}_{1:t}$ adalah $O(t)$. Ini adalah kompleksitas untuk penghalusan pada langkah waktu tertentu k . Jika kita ingin menghaluskan seluruh urutan, satu metode yang jelas adalah dengan menjalankan seluruh proses penghalusan sekali untuk setiap langkah waktu yang akan dihaluskan. Hal ini menghasilkan kompleksitas waktu $O(t^2)$. Pendekatan yang lebih baik menggunakan aplikasi sederhana pemrograman dinamis untuk mengurangi kompleksitas menjadi $O(t)$. Petunjuk muncul dalam analisis sebelumnya dari contoh payung, di mana kita dapat menggunakan kembali hasil dari fase penyaringan maju. Kunci dari algoritma waktu-linear adalah *mencatat hasil* penyaringan maju pada seluruh urutan. Kemudian kita menjalankan rekursi mundur dari t ke bawah hingga 1, menghitung estimasi yang dihaluskan pada setiap langkah k dari pesan mundur yang dihitung $\mathbf{b}_{k+1:t}$ dan pesan maju yang disimpan $\mathbf{f}_{1:k}$. Algoritma tersebut, yang disebut **algoritma maju-mundur**, ditunjukkan pada Gambar 6.4.

Pembaca yang waspada akan melihat bahwa struktur jaringan Bayesian yang ditunjukkan pada Gambar 6.3 adalah *polytree* sebagaimana didefinisikan pada halaman

sebelumnya. Ini berarti bahwa penerapan langsung dari algoritma pengelompokan juga menghasilkan algoritma waktu-linear yang menghitung estimasi yang dihaluskan untuk seluruh urutan. Sekarang dipahami bahwa algoritma maju-mundur sebenarnya adalah kasus khusus dari algoritma propagasi polytree yang digunakan dengan metode pengelompokan (meskipun keduanya dikembangkan secara independen).

Algoritma maju-mundur membentuk tulang punggung komputasi untuk banyak aplikasi yang menangani urutan pengamatan yang bising. Seperti yang dijelaskan sejauh ini, algoritma ini memiliki dua kelemahan praktis. Yang pertama adalah kompleksitas ruangnya bisa terlalu tinggi ketika ruang statusnya besar dan urutannya panjang. Algoritma ini menggunakan ruang $O(|\mathbf{f}|t)$ di mana $|\mathbf{f}|$ adalah ukuran representasi pesan penerusan. Persyaratan ruang dapat dikurangi menjadi $O(|\mathbf{f}|\log t)$ dengan peningkatan kompleksitas waktu secara bersamaan dengan faktor $\log t$,

Kelemahan kedua dari algoritma dasar ini adalah algoritma ini perlu dimodifikasi agar dapat bekerja dalam pengaturan *daring* di mana estimasi yang dihaluskan harus dihitung untuk irisan waktu sebelumnya karena pengamatan baru terus ditambahkan ke akhir urutan. Persyaratan yang paling umum adalah untuk **penghalusan jeda tetap**, yang memerlukan penghitungan estimasi yang dihaluskan $\mathbf{P}(\mathbf{X}_{t-d}|\mathbf{e}_{1:t})$ untuk d tetap. Artinya, penghalusan dilakukan untuk irisan waktu d langkah di belakang waktu t saat ini; saat t meningkat, penghalusan harus mengikutinya. Jelas, kita dapat menjalankan algoritma maju-mundur pada "jendela" d -step saat setiap pengamatan baru ditambahkan, tetapi ini tampaknya tidak efisien. Di Bagian 15.3, kita akan melihat bahwa penghalusan kelambatan tetap dapat, dalam beberapa kasus, dilakukan dalam waktu konstan per pembaruan, terlepas dari kelambatan d .

Menemukan urutan yang paling mungkin

Misalkan $[true, true, false, true, true]$ adalah urutan payung untuk lima hari pertama petugas keamanan bekerja. Urutan cuaca apa yang paling mungkin menjelaskan hal ini? Apakah tidak adanya payung pada hari ke-3 berarti tidak hujan, atau apakah direktur lupa membawanya? Jika tidak hujan pada hari ke-3, mungkin (karena cuaca cenderung bertahan) tidak hujan juga pada hari ke-4, tetapi direktur membawa payung untuk berjaga-jaga. Secara keseluruhan, ada 2^5 kemungkinan urutan cuaca yang dapat kita pilih. Apakah ada cara untuk menemukan yang paling mungkin, selain menghitung semuanya?

Kita dapat mencoba prosedur waktu linier ini: gunakan penghalusan untuk menemukan distribusi posterior untuk cuaca pada setiap langkah waktu; kemudian buat urutannya, dengan menggunakan pada setiap langkah cuaca yang paling mungkin menurut posterior. Pendekatan semacam itu seharusnya membunyikan lonceng peringatan di benak pembaca, karena distribusi posterior yang dihitung dengan penghalusan adalah distribusi selama langkah waktu *tunggal*, sedangkan untuk menemukan *urutan* yang paling mungkin, kita harus mempertimbangkan probabilitas *gabungan* selama semua langkah waktu. Hasilnya sebenarnya bisa sangat berbeda.

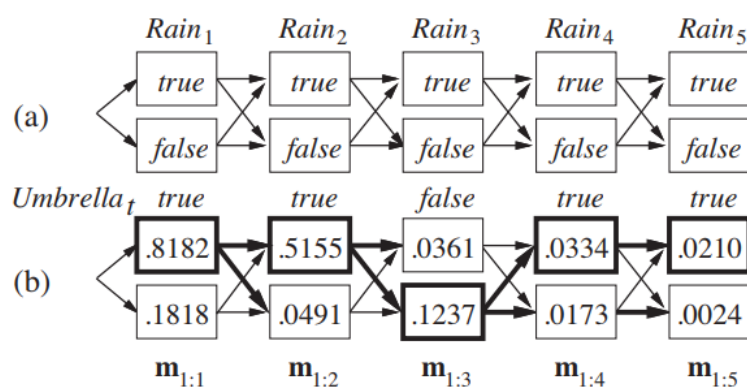
```

function FORWARD – BACKWARD(ev, prior) returns a vector of probability distributions
  inputs: ev, a vector of evidence values for steps 1, ..., t
           prior, the prior distribution on the initial state,  $P(X_0)$ 
  local variables: fv, a vector of forward messages for steps 0, ..., t
                    b, a representation of the backward message, initially all 1s
                    sv, a vector of smoothed estimates for steps 1, ..., t

  fv[0]  $\leftarrow$  prior
  for i = 1 to t do
    fv[i]  $\leftarrow$  FORWARD(fv[i - 1], ev[i])
  for i = t downto 1 do
    sv[i]  $\leftarrow$  NORMALIZE(fv[i]  $\times$  b)
    b  $\leftarrow$  BACKWARD(b, ev[i])
  return sv

```

Gambar 6.4 Algoritma maju-mundur untuk penghalusan: menghitung probabilitas posterior dari serangkaian keadaan yang diberikan serangkaian pengamatan. Operator MAJU dan BELOK didefinisikan oleh Persamaan (6.5) dan (6.9), masing-masing.



Gambar 6.5 (a) Urutan keadaan yang mungkin untuk $Rain_t$ dapat dilihat sebagai lintasan melalui grafik keadaan yang mungkin pada setiap langkah waktu. (Keadaan ditampilkan sebagai persegi panjang untuk menghindari kebingungan dengan simpul dalam jaringan Bayes.) (b) Pengoperasian algoritma Viterbi untuk urutan observasi payung $[true, true, false, true, true]$. Untuk setiap t , kami telah menunjukkan nilai pesan $m_{1:t}$, yang memberikan probabilitas urutan terbaik yang mencapai setiap keadaan pada waktu t . Selain itu, untuk setiap keadaan, panah tebal yang mengarah ke sana menunjukkan pendahulunya yang terbaik sebagaimana diukur dengan produk probabilitas urutan sebelumnya dan probabilitas transisi. Mengikuti panah tebal kembali dari keadaan yang paling mungkin dalam $m_{1:5}$ memberikan urutan yang paling mungkin.

Ada algoritme waktu linier untuk menemukan urutan yang paling mungkin, tetapi memerlukan sedikit pemikiran lebih lanjut. Algoritme ini bergantung pada properti Markov yang sama yang menghasilkan algoritme yang efisien untuk penyaringan dan penghalusan. Cara termudah untuk memikirkan masalah ini adalah dengan melihat setiap urutan sebagai *jalur* melalui grafi yang simpulnya adalah kemungkinan *keadaan* pada setiap langkah waktu.

Grafik seperti itu ditunjukkan untuk dunia payung pada Gambar 6.5(a). Sekarang pertimbangkan tugas untuk menemukan jalur yang paling mungkin melalui grafik ini, di mana kemungkinan jalur apa pun adalah produk dari probabilitas transisi di sepanjang jalur dan probabilitas pengamatan yang diberikan pada setiap keadaan. Mari kita fokus secara khusus pada jalur yang mencapai keadaan $Rain_5 = true$. Karena sifat Markov, maka dapat disimpulkan bahwa jalur yang paling mungkin menuju keadaan $Rain_5 = true$ terdiri dari jalur yang paling mungkin menuju *beberapa* keadaan pada waktu 4 diikuti oleh transisi ke $Rain_5 = true$; dan keadaan pada waktu 4 yang akan menjadi bagian dari jalur menuju $Rain_5 = true$ adalah yang memaksimalkan kemungkinan jalur tersebut. Dengan kata lain, *terdapat hubungan rekursif antara jalur yang paling mungkin menuju setiap keadaan x_{t+1} dan jalur yang paling mungkin menuju setiap keadaan x_t* . Kita dapat menuliskan hubungan ini sebagai persamaan yang menghubungkan probabilitas jalur:

$$\begin{aligned} & \max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} | e_{1:t+1}) \\ &= \alpha P(e_{t+1} | X_{t+1}) \max_{x_t} \left(P(X_{t+1} | x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t | e_{1:t}) \right). \end{aligned} \quad (6.11)$$

Persamaan (6.11) *identik* dengan persamaan penyaringan (6.5) kecuali bahwa

1. Pesan penerusan $f_{1:t} = P(x_t | e_{1:t})$ digantikan oleh pesan yaitu,

$$m_{1:t} = \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, X_t | e_{1:t}),$$

probabilitas jalur yang paling mungkin menuju setiap keadaan x_t ; dan

2. penjumlahan atas x_t pada Persamaan (6.5) digantikan dengan pemaksimalan atas x_t pada Persamaan (6.11).

Dengan demikian, algoritme untuk menghitung urutan yang paling mungkin mirip dengan penyaringan: ia berjalan maju sepanjang urutan, menghitung pesan m pada setiap langkah waktu, menggunakan Persamaan (6.11). Kemajuan perhitungan ini ditunjukkan pada Gambar 6.5(b). Pada akhirnya, ia akan memiliki probabilitas untuk urutan yang paling mungkin mencapai *setiap* status akhir. Dengan demikian, seseorang dapat dengan mudah memilih urutan yang paling mungkin secara keseluruhan (status yang digarisbawahi dengan huruf tebal). Untuk mengidentifikasi urutan yang sebenarnya, sebagai lawan dari hanya menghitung probabilitasnya, algoritme juga perlu mencatat, untuk setiap status, status terbaik yang mengarah padanya; ini ditunjukkan oleh panah tebal pada Gambar 6.5(b). Urutan optimal diidentifikasi dengan mengikuti panah tebal ini mundur dari status akhir terbaik.

Algoritme yang baru saja kami jelaskan disebut **algoritme Viterbi**, sesuai dengan nama penemunya. Seperti algoritme penyaringan, kompleksitas waktunya linier dalam t , panjang urutan. Tidak seperti penyaringan, yang menggunakan ruang konstan, kebutuhan ruangnya juga linier dalam t . Hal ini karena algoritma Viterbi perlu menyimpan petunjuk yang mengidentifikasi urutan terbaik yang mengarah ke setiap status.

6.3 MODEL MARKOV TERSEMBUNYI

Bagian sebelumnya mengembangkan algoritme untuk penalaran probabilistik temporal menggunakan kerangka umum yang independen dari bentuk khusus model transisi dan sensor. Dalam bagian ini dan dua bagian berikutnya, kami membahas model dan aplikasi yang lebih konkret yang menggambarkan kekuatan algoritme dasar dan dalam beberapa kasus memungkinkan peningkatan lebih lanjut.

Kami mulai dengan **model Markov tersembunyi**, atau **HMM**. HMM adalah model probabilistik temporal di mana status proses dijelaskan oleh *satu variabel acak diskret*. Nilai variabel yang mungkin adalah status dunia yang mungkin. Oleh karena itu, contoh umum yang dijelaskan di bagian sebelumnya adalah HMM, karena hanya memiliki satu variabel status: $Rain_t$. Apa yang terjadi jika Anda memiliki model dengan dua atau lebih variabel status? Anda masih dapat memasukkannya ke dalam kerangka HMM dengan menggabungkan variabel menjadi satu "megavariabel" yang nilainya adalah semua tupel nilai variabel status individual yang mungkin. Kita akan melihat bahwa struktur terbatas HMM memungkinkan implementasi matriks yang sederhana dan elegan dari semua algoritma dasar.

Algoritma matriks yang disederhanakan

Dengan satu variabel status diskret X_t , kita dapat memberikan bentuk konkret pada representasi model transisi, model sensor, dan pesan maju dan mundur. Biarkan variabel status X_t memiliki nilai yang dilambangkan dengan bilangan bulat $1, \dots, S$, di mana S adalah jumlah status yang mungkin. Model transisi $P(X_t | X_{t-1})$ menjadi $S \times S$ matriks $\mathbf{T}$, di mana

$$\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i).$$

Yaitu, $\mathbf{T}_{ij}$ adalah probabilitas transisi dari keadaan i ke keadaan j . Misalnya, matriks transisi untuk dunia payung adalah

$$\mathbf{T} = P(X_t | X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}.$$

Kami juga menempatkan model sensor dalam bentuk matriks. Dalam kasus ini, karena nilai variabel bukti E_t diketahui pada waktu t (sebut saja e_t), kami hanya perlu menentukan, untuk setiap keadaan, seberapa besar kemungkinan keadaan tersebut menyebabkan e_t muncul: kami memerlukan $P(e_t | X_t = i)$ untuk setiap keadaan i . Untuk kemudahan matematika, kami menempatkan nilai-nilai ini ke dalam matriks diagonal $S \times S$, $\mathbf{O}_t$ yang entri diagonal ke- i -nya adalah $P(e_t | X_t = i)$ dan entri lainnya adalah 0. Misalnya, pada hari ke-1 di dunia payung Gambar 6.5, $U_1 = \text{true}$, dan pada hari ke-3, $U_3 = \text{false}$, jadi, dari Gambar 6.2, kami memiliki

$$\mathbf{o}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}; \quad \mathbf{o}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}.$$

Sekarang, jika kita menggunakan vektor kolom untuk mewakili pesan maju dan mundur, semua perhitungan menjadi operasi matriks-vektor sederhana. Persamaan maju (6.5) menjadi

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^T \mathbf{f}_{1:t} \quad (6.12)$$

dan persamaan mundur (15.9) menjadi

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}. \quad (6.13)$$

Dari persamaan ini, kita dapat melihat bahwa kompleksitas waktu dari algoritma maju-mundur (Gambar 6.4) yang diterapkan pada urutan dengan panjang t adalah $O(S^2 t)$, karena setiap langkah memerlukan perkalian vektor elemen S dengan matriks $S \times S$. Persyaratan ruang adalah $O(St)$, karena lintasan maju menyimpan t vektor berukuran S .

Selain memberikan deskripsi yang elegan tentang algoritma penyaringan dan penghalusan untuk HMM, formulasi matriks mengungkapkan peluang untuk algoritma yang lebih baik. Yang pertama adalah variasi sederhana pada algoritma maju-mundur yang memungkinkan penghalusan dilakukan dalam ruang *konstan*, terlepas dari panjang urutan. Idenya adalah bahwa penghalusan untuk setiap irisan waktu tertentu k memerlukan kehadiran simultan dari pesan maju dan mundur, $\mathbf{f}_{1:k}$ dan $\mathbf{b}_{k+1:t}$, menurut Persamaan (6.8). Algoritma maju-mundur mencapai hal ini dengan menyimpan $\mathbf{f}_s$ yang dihitung pada lintasan maju sehingga tersedia selama lintasan mundur. Cara lain untuk mencapai hal ini adalah dengan satu lintasan yang menyebarkan $\mathbf{f}$ dan $\mathbf{b}$ ke arah yang sama. Misalnya, pesan “maju” $\mathbf{f}$ dapat disebarkan ke belakang jika kita memanipulasi Persamaan (6.12) untuk bekerja ke arah yang lain:

$$\mathbf{f}_{1:t} = \alpha' (\mathbf{T}^T)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1}.$$

Algoritma penghalusan yang dimodifikasi bekerja dengan terlebih dahulu menjalankan forward pass standar untuk menghitung $\mathbf{f}_{t:t}$ (melupakan semua hasil antara) dan kemudian menjalankan backward pass untuk $\mathbf{b}$ dan $\mathbf{f}$ secara bersamaan, menggunakannya untuk menghitung estimasi yang dihaluskan pada setiap langkah. Karena hanya diperlukan satu salinan dari setiap pesan, persyaratan penyimpanan bersifat konstan (yaitu, tidak bergantung pada t , panjang urutan). Ada dua batasan signifikan pada algoritma ini: ia mensyaratkan bahwa matriks transisi dapat dibalik dan bahwa model sensor tidak memiliki nol—yaitu, bahwa setiap pengamatan dimungkinkan dalam setiap status.

```
function FIXED-LAG-SMOOTHING( $e_t, hmm, d$ ) returns a distribution over  $\mathbf{X}_{t-d}$ 
  inputs:  $e_t$ , the current evidence for time step  $t$ 
          $hmm$ , a hidden Markov model with  $S \times S$  transition matrix  $\mathbf{T}$ 
          $d$ , the length of the lag for smoothing
  persistent:  $t$ , the current time, initially 1
               $\mathbf{f}$ , the forward message  $P(\mathbf{X}_t | \mathbf{e}_{1:t})$ , initially  $hmm.PRIOR$ 
               $\mathbf{B}$ , the  $d$ -step backward transformation matrix, initially the identity matrix
               $e_{t-d:t}$ , double-ended list of evidence from  $t-d$  to  $t$ , initially empty
```

```

local variables:  $O_{t-d}$ ,  $O_t$ , diagonal matrices containing the sensor model information

add  $e_t$  to the end of  $e_{t-d:t}$ 
 $O_t \leftarrow$  diagonal matrix containing  $P(e_t | X_t)$ 
if  $t > d$  then
     $f \leftarrow \text{FORWARD}(f, e_t)$ 
    remove  $e_{t-d-1}$  from the beginning of  $e_{t-d:t}$ 
     $O_{t-d} \leftarrow$  diagonal matrix containing  $P(e_{t-d} | X_{t-d})$ 
     $B \leftarrow O_{t-d}^{-1} T^{-1} B T O_t$ 
else  $B \leftarrow B T O_t$ 
 $t \leftarrow t + 1$ 
if  $t > d$  then return  $\text{NORMALIZE}(f \times B1)$  else return null

```

```

function FIXED – LAG – SMOOTHING( $e_t, hmm, d$ ) returns a distribution over  $X_{t-d}$ 
inputs:  $e_t$ , the current evidence for time step  $t$ 
          $hmm$ , a hidden Markov model with  $S \times S$  transition matrix  $T$ 
          $d$ , the length of the lag for smoothing
persistent:  $t$ , the current time, initially 1
               $f$ , the forward message  $P(X_t | e_{1:t})$ , initially  $hmm.PRIOR$ 
               $B$ , the  $d$  – step backward transformation matrix, initially the identity matrix
               $e_{t-d:t}$ , double – ended list of evidence from  $t - d$  to  $t$ , initially empty
local variables:  $O_{t-d}$ ,  $O_t$ , diagonal matrices containing the sensor model information

add  $e_t$  to the end of  $e_{t-d:t}$ 
 $O_t \leftarrow$  diagonal matrix containing  $P(e_t | X_t)$ 
if  $t > d$  then
     $f \leftarrow \text{FORWARD}(f, e_t)$ 
    remove  $e_{t-d-1}$  from the beginning of  $e_{t-d:t}$ 
     $O_{t-d} \leftarrow$  diagonal matrix containing  $P(e_{t-d} | X_{t-d})$ 
     $B \leftarrow O_{t-d}^{-1} B T O_t$ 
else  $B \leftarrow B T O_t$ 
 $t \leftarrow t + 1$ 
if  $t > d$  then return  $\text{NORMALIZE}(f \times B1)$  else return null

```

Gambar 6.6 Algoritma untuk penghalusan dengan jeda waktu tetap d langkah, diimplementasikan sebagai algoritma daring yang menghasilkan estimasi penghalusan baru berdasarkan pengamatan untuk langkah waktu baru. Perhatikan bahwa keluaran akhir $\text{NORMALIZE}(f \times B1)$ hanya $\alpha f \times b$, menurut Persamaan (6.14).

Area kedua di mana formulasi matriks menunjukkan peningkatan adalah dalam penghalusan *daring* dengan jeda tetap. Fakta bahwa penghalusan dapat dilakukan dalam ruang konstan menunjukkan bahwa harus ada algoritma rekursif yang efisien untuk penghalusan daring—yaitu, algoritma yang kompleksitas waktunya tidak bergantung pada panjang jeda. Mari kita asumsikan bahwa jeda adalah d ; yaitu, kita melakukan penghalusan

pada irisan waktu $t - d$, di mana waktu saat ini adalah t . Dengan Persamaan (6.8), kita perlu menghitung

$$\alpha \mathbf{f}_{1:t-d} \times \mathbf{b}_{t-d+1:t}$$

untuk irisan $t - d$. Kemudian, ketika observasi baru tiba, kita perlu menghitung

$$\alpha \mathbf{f}_{1:t-d+1} \times \mathbf{b}_{t-d+2:t+1}$$

untuk irisan $t - d + 1$. Bagaimana ini dapat dilakukan secara bertahap? Pertama, kita dapat menghitung $\mathbf{f}_{1:t-d+1}$ dari $\mathbf{f}_{1:t-d}$, menggunakan proses penyaringan standar, Persamaan (6.5).

Menghitung pesan mundur secara bertahap lebih sulit, karena tidak ada hubungan sederhana antara pesan mundur lama $\mathbf{b}_{t-d+1:t}$ dan pesan mundur baru $\mathbf{b}_{t-d+2:t+1}$. Sebaliknya, kita akan memeriksa hubungan antara pesan mundur lama $\mathbf{b}_{t-d+1:t}$ dan pesan mundur di bagian depan urutan, $\mathbf{b}_{t+1:t}$. Untuk melakukan ini, kita menerapkan Persamaan (6.13) sebanyak d kali untuk mendapatkan

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^t \mathbf{TO}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t} \mathbf{1}, \quad (6.14)$$

di mana matriks $\mathbf{b}_{t-d+1:t}$ adalah hasil perkalian dari urutan matriks $\mathbf{T}$ dan $\mathbf{O}$. $\mathbf{B}$ dapat dianggap sebagai “operator transformasi” yang mengubah pesan mundur berikutnya menjadi pesan sebelumnya. Persamaan serupa berlaku untuk pesan mundur baru *setelah* observasi berikutnya tiba:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=t-d+2}^{t+1} \mathbf{TO}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1} \mathbf{1}. \quad (6.15)$$

Dengan memeriksa ekspresi produk dalam Persamaan (6.14) dan (15.15), kita melihat bahwa keduanya memiliki hubungan sederhana: untuk mendapatkan produk kedua, “bagi” produk pertama dengan elemen pertama $\mathbf{TO}_{t-d+1}$, dan kalikan dengan elemen terakhir yang baru $\mathbf{TO}_{t+1}$. Dalam bahasa matriks, maka, terdapat hubungan sederhana antara matriks $\mathbf{B}$ lama dan baru:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1} \mathbf{T}^{-1} \mathbf{B}_{t-d+1:t} \mathbf{TO}_{t+1}. \quad (6.16)$$

Persamaan ini memberikan pembaruan tambahan untuk matriks $\mathbf{B}$, yang pada gilirannya (melalui Persamaan (6.15)) memungkinkan kita menghitung pesan mundur baru $\mathbf{b}_{t-d+2:t+1}$. Algoritme lengkap, yang memerlukan penyimpanan dan pembaruan $\mathbf{f}$ dan $\mathbf{B}$, ditunjukkan pada Gambar 6.6.

Contoh model Markov Tersembunyi: Lokalisasi

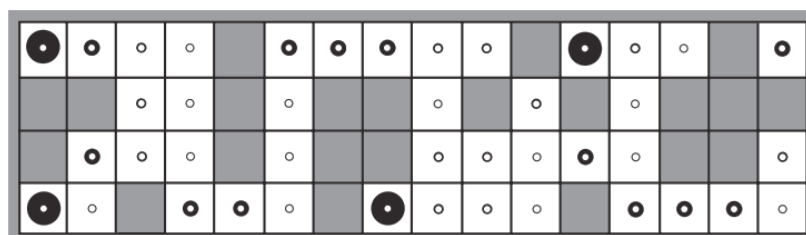
Pada halaman sebelumnya, kami memperkenalkan bentuk sederhana dari masalah **lokalisasi** untuk dunia vakum. Dalam versi tersebut, robot memiliki satu tindakan *Bergerak* nondeterministik dan sensornya melaporkan dengan sempurna apakah rintangan berada tepat di utara, selatan, timur, dan barat; status keyakinan robot adalah serangkaian kemungkinan lokasi tempat robot berada.

Di sini kami membuat masalah sedikit lebih realistis dengan menyertakan model probabilitas sederhana untuk gerakan robot dan dengan memperhitungkan derau pada sensor. Variabel status X_t merepresentasikan lokasi robot pada kisi diskret; domain variabel ini adalah serangkaian kotak kosong $\{s_1, \dots, s_n\}$. Misalkan $\text{NEIGHBORS}(s)$ adalah himpunan kotak kosong yang berdekatan dengan s dan misalkan $N(s)$ adalah ukuran himpunan tersebut. Maka model transisi untuk tindakan *Move* menyatakan bahwa robot memiliki kemungkinan yang sama untuk berakhir di kotak tetangga mana pun:

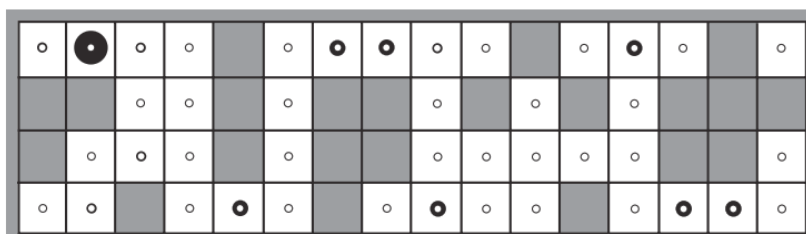
$$P(X_{t+1} = j \mid X_t = i) = \mathbf{T}_{ij} = (1/N(i) \text{ if } j \in \text{Neighbors}(i) \text{ else } 0).$$

Kita tidak tahu di mana robot itu mulai, jadi kita akan mengasumsikan distribusi seragam di semua kotak; yaitu, $P(X_0 = i) = 1/n$. Untuk lingkungan tertentu yang kita pertimbangkan (Gambar 6.7), $n = 42$ dan matriks transisi $\mathbf{T}$ memiliki $42 \times 42 = 1764$ entri.

Variabel sensor E_t memiliki 16 kemungkinan nilai, masing-masing berupa urutan empat bit yang menunjukkan ada atau tidaknya halangan dalam arah kompas tertentu. Kita akan menggunakan notasi NS , misalnya, untuk mengartikan bahwa sensor utara dan selatan melaporkan halangan dan timur dan barat tidak.



(a) Posterior distribution over robot location after $E_1 = NSW$



(b) Posterior distribution over robot location after $E_1 = NSW, E_2 = NS$

Gambar 6.7 Distribusi posterior di lokasi robot: (a) satu pengamatan $E_1 = NSW$; (b) setelah pengamatan kedua $E_2 = NS$. Ukuran setiap cakram sesuai dengan probabilitas bahwa robot berada di lokasi tersebut. Tingkat kesalahan sensor adalah $\epsilon = 0,2$.

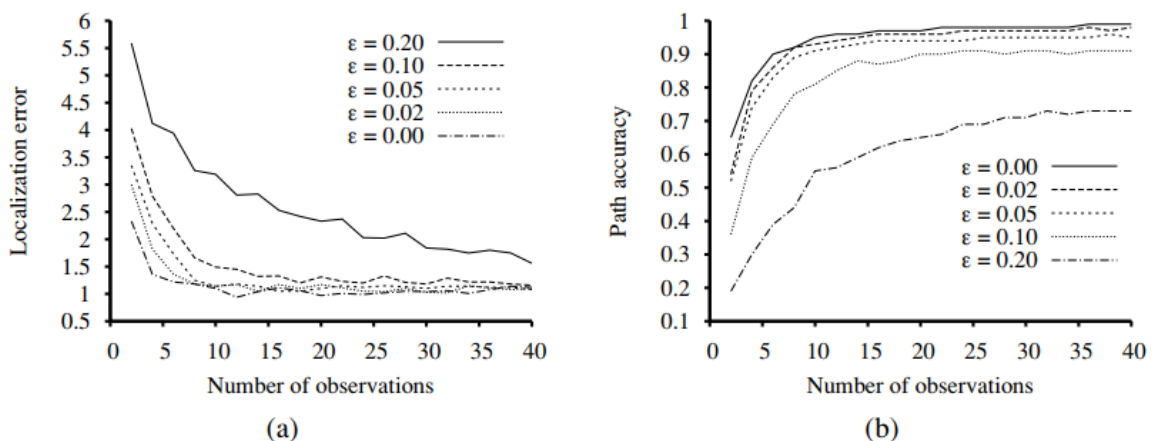
Misalkan tingkat kesalahan setiap sensor adalah ϵ dan kesalahan terjadi secara independen untuk keempat arah sensor. Dalam hal itu, probabilitas mendapatkan keempat bit dengan benar adalah $(1 - \epsilon)^4$ dan probabilitas mendapatkan semuanya salah adalah ϵ^4 . Lebih jauh lagi, jika d_{it} adalah perbedaan—jumlah bit yang berbeda—antara nilai sebenarnya untuk kotak i dan pembacaan aktual e_t , maka probabilitas bahwa robot di kotak i akan menerima pembacaan sensor e_t adalah

$$P(E_t = e_t | X_t = i) = \mathbf{O}_{ti} = (1 - \epsilon)^{4-d_{it}} \epsilon^{d_{it}}.$$

Misalnya, probabilitas bahwa sebuah kotak dengan rintangan di utara dan selatan akan menghasilkan pembacaan sensor NSE adalah $(1 - \epsilon)^3 \epsilon^1$.

Dengan matriks $\mathbf{T}$ dan $\mathbf{O}_t$, robot dapat menggunakan Persamaan (6.12) untuk menghitung distribusi posterior di lokasi—yaitu, untuk mencari tahu di mana robot berada. Gambar 6.7 menunjukkan distribusi $\mathbf{P}(X_1 | E_1 = NSW)$ dan $\mathbf{P}(X_2 | E_1 = NSW, E_2 = NS)$. Ini adalah labirin yang sama yang kita lihat sebelumnya, tetapi di sana kita menggunakan penyaringan logis untuk menemukan lokasi yang *memungkinkan*, dengan asumsi penginderaan yang sempurna. Lokasi yang sama tersebut masih merupakan yang *paling mungkin* dengan penginderaan yang bising, tetapi sekarang *setiap* lokasi memiliki beberapa probabilitas bukan nol.

Selain penyaringan untuk memperkirakan lokasinya saat ini, robot dapat menggunakan penghalusan (Persamaan (6.13)) untuk mengetahui di mana ia berada pada waktu lampau tertentu—misalnya, di mana ia mulai pada waktu 0—dan ia dapat menggunakan algoritme Viterbi untuk mengetahui jalur yang paling mungkin telah diambilnya untuk sampai ke tempatnya sekarang. Gambar 6.8 menunjukkan kesalahan lokalisasi dan akurasi jalur Viterbi untuk berbagai nilai rasio kesalahan sensor per bit ϵ .



Gambar 6.8 Kinerja lokalisasi HMM sebagai fungsi panjang urutan observasi untuk berbagai nilai probabilitas kesalahan sensor ϵ yang berbeda; data dirata-ratakan selama 400 kali pengujian. (a) Kesalahan lokalisasi, didefinisikan sebagai jarak Manhattan dari

lokasi sebenarnya. (b) Akurasi lintasan Viterbi, didefinisikan sebagai fraksi status yang benar pada lintasan Viterbi.

Bahkan ketika ϵ adalah 20%—yang berarti bahwa pembacaan sensor keseluruhan salah 59% dari waktu—robot biasanya dapat mengetahui lokasinya dalam dua kotak setelah 25 pengamatan. Ini karena kemampuan algoritme untuk mengintegrasikan bukti dari waktu ke waktu dan memperhitungkan kendala probabilistik yang dikenakan pada urutan lokasi oleh model transisi. Ketika ϵ adalah 10%, kinerja setelah setengah lusin pengamatan sulit dibedakan dari kinerja dengan penginderaan sempurna. Secara umum, tingkat pelokalan dan akurasi jalur yang tinggi dipertahankan bahkan saat menghadapi kesalahan substansial dalam model yang digunakan.

Variabel status untuk contoh yang telah kita bahas di bagian ini adalah lokasi fisik di dunia. Masalah lain tentu saja dapat mencakup aspek lain di dunia. Latihan 15.8 meminta Anda untuk mempertimbangkan versi robot vakum yang memiliki kebijakan untuk bergerak lurus selama mungkin; hanya ketika robot tersebut menemui rintangan, robot tersebut berubah ke arah baru (dipilih secara acak). Untuk memodelkan robot ini, setiap status dalam model terdiri dari pasangan (*lokasi*, *arah*). Untuk lingkungan pada Gambar 6.7, yang memiliki 42 kotak kosong, hal ini menghasilkan 168 status dan matriks transisi dengan $168^2 = 28,224$ entri—masih merupakan angka yang dapat dikelola. Jika kita menambahkan kemungkinan adanya kotoran di kotak-kotak tersebut, jumlah status dikalikan dengan 2^{42} dan matriks transisi berakhir dengan lebih dari 10^{29} entri—bukan lagi angka yang dapat dikelola; Bagian selanjutnya menunjukkan cara menggunakan jaringan Bayesian dinamis untuk memodelkan domain dengan banyak variabel status. Jika kita membiarkan robot bergerak terus-menerus daripada dalam jaringan diskret, jumlah statusnya menjadi tak terbatas; bagian berikutnya menunjukkan cara menangani kasus ini.

6.4 FILTER KALMAN

Bayangkan sedang mengamati seekor burung kecil terbang di antara dedaunan hutan lebat saat senja: Anda melihat sekilas gerakan yang singkat dan terputus-putus; Anda berusaha keras menebak di mana burung itu berada dan di mana ia akan muncul berikutnya agar Anda tidak kehilangannya. Atau bayangkan Anda adalah operator radar Perang Dunia II yang sedang mengamati titik samar yang muncul setiap 10 detik di layar. Atau, jika melihat lebih jauh ke belakang, bayangkan Anda adalah Kepler yang mencoba merekonstruksi gerakan planet-planet dari kumpulan pengamatan sudut yang sangat tidak akurat yang diambil pada interval yang tidak teratur dan tidak diukur dengan tepat.

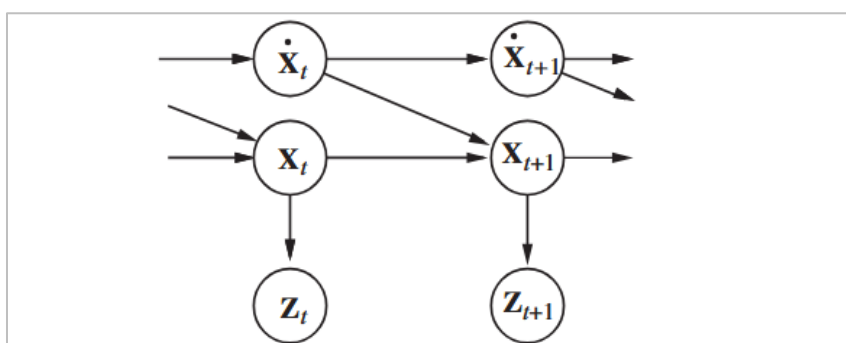
Dalam semua kasus ini, Anda melakukan penyaringan: memperkirakan variabel keadaan (di sini, posisi dan kecepatan) dari pengamatan yang bising dari waktu ke waktu. Jika variabelnya diskrit, kita dapat memodelkan sistem dengan model Markov tersembunyi. Bagian ini membahas metode untuk menangani variabel kontinu, menggunakan algoritma yang disebut penyaringan Kalman, yang diambil dari salah satu penemunya, Rudolf E. Kalman.

Penerbangan burung dapat ditentukan oleh enam variabel kontinu pada setiap titik waktu; tiga untuk posisi (X_t, Y_t, Z_t) dan tiga untuk kecepatan ($\dot{X}_t, \dot{Y}_t, \dot{Z}_t$).

Kita akan memerlukan kerapatan kondisional yang sesuai untuk merepresentasikan model transisi dan sensor, kita akan menggunakan distribusi Gaussian linear. Ini berarti bahwa keadaan selanjutnya X_{t+1} harus merupakan fungsi linear dari keadaan saat ini X_t , ditambah beberapa derau Gaussian, suatu kondisi yang ternyata cukup masuk akal dalam praktik. Pertimbangkan, misalnya, koordinat X burung, abaikan koordinat lainnya untuk saat ini. Biarkan interval waktu antara pengamatan menjadi Δ , dan asumsikan kecepatan konstan selama interval; maka pembaruan posisi diberikan oleh $X_{t+\Delta} = X_t + \dot{X}_t \Delta$. Dengan menambahkan derau Gaussian (untuk memperhitungkan variasi angin, dll.), kita memperoleh model transisi Gaussian linear:

$$P(X_{t+\Delta} = x_{t+\Delta} \mid X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t \Delta, \sigma^2)(x_{t+\Delta})$$

Struktur jaringan Bayesian untuk sistem dengan vektor posisi X_t dan kecepatan $\dot{X}_t$ ditunjukkan pada Gambar 6.9. Perhatikan bahwa ini adalah bentuk model Gaussian linear yang sangat spesifik; bentuk umumnya akan dijelaskan nanti di bagian ini dan mencakup berbagai macam aplikasi di luar contoh gerakan sederhana dari paragraf pertama.



Gambar 6.9 Struktur jaringan Bayesian untuk sistem dinamis linier dengan posisi X_t , kecepatan $\dot{X}_t$, dan pengukuran posisi Z_t .

Memperbarui Distribusi Gaussian

Kami menyinggung properti utama dari keluarga distribusi Gaussian linear: ia tetap tertutup di bawah operasi jaringan Bayesian standar. Di sini, kami membuat klaim ini tepat dalam konteks penyaringan dalam model probabilitas temporal. Properti yang dibutuhkan sesuai dengan perhitungan penyaringan dua langkah dalam Persamaan (6.5):

1. Jika distribusi saat ini $P(X_t \mid e_{1:t})$ adalah Gaussian dan model transisi $P(X_{t+1} \mid x_t)$ adalah Gaussian linier, maka distribusi prediksi satu langkah yang diberikan oleh

$$P(X_{t+1} \mid e_{1:t}) = \int_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid e_{1:t}) dx_t \quad (6.17)$$

juga merupakan distribusi Gaussian.

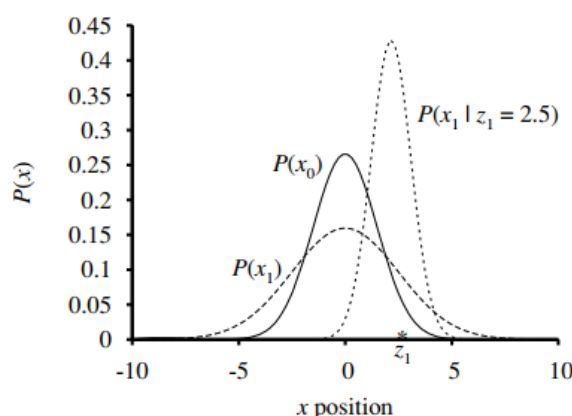
2. Jika prediksi $P(X_{t+1} | e_{1:t})$ adalah Gaussian dan model sensor $P(e_{t+1} | X_{t+1})$ adalah Gaussian linier, maka, setelah pengkondisian pada bukti baru, distribusi yang diperbarui

$$P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t}) \quad (6.18)$$

juga merupakan distribusi Gaussian.

Jadi, operator *FORWARD* untuk penyaringan Kalman mengambil pesan penerusan Gaussian $f_{1:t}$, yang ditetapkan oleh mean μ_t dan matriks kovariansi Σ_t , dan menghasilkan pesan penerusan Gaussian multivariat baru $f_{1:t+1}$, yang ditetapkan oleh mean μ_{t+1} dan matriks kovariansi Σ_{t+1} . Jadi, jika kita mulai dengan prior Gaussian $f_{1:0} = P(X_0) = N(\mu_0, \Sigma_0)$, penyaringan dengan model Gaussian linear menghasilkan distribusi status Gaussian untuk semua waktu.

Ini tampaknya merupakan hasil yang bagus dan elegan, tetapi mengapa ini begitu penting? Alasannya adalah, kecuali untuk beberapa kasus khusus seperti ini, penyaringan dengan jaringan kontinu atau hibrida (diskrit dan kontinu) menghasilkan distribusi status yang representasinya tumbuh tanpa batas seiring waktu. Pernyataan ini tidak mudah dibuktikan secara umum.



Gambar 6.10 Tahapan dalam siklus pembaruan filter Kalman untuk random walk dengan prior yang diberikan oleh $\mu_0 = 0,0$ dan $\sigma_0 = 1,0$, transition noise yang diberikan oleh $\sigma_x = 2,0$, sensor noise yang diberikan oleh $\sigma_z = 1,0$, dan observasi pertama $z_1 = 2,5$ (ditandai pada sumbu x). Perhatikan bagaimana prediksi $P(x_1)$ diratakan, relatif terhadap $P(x_0)$, oleh transition noise. Perhatikan juga bahwa mean dari distribusi posterior $P(x_1 | z_1)$ sedikit ke kiri dari observasi z_1 karena mean adalah rata-rata tertimbang dari prediksi dan observasi.

Contoh Satu Dimensi yang Sederhana

Kami telah mengatakan bahwa operator *FORWARD* untuk filter Kalman memetakan Gaussian ke Gaussian baru. Ini berarti menghitung matriks rata-rata dan kovariansi baru dari matriks rata-rata dan kovariansi sebelumnya. Menurunkan aturan pembaruan dalam kasus umum (multivariat) memerlukan banyak aljabar linier, jadi kami akan tetap menggunakan kasus univariat yang sangat sederhana untuk saat ini; dan kemudian memberikan hasil untuk kasus umum. Bahkan untuk kasus univariat, perhitungannya agak membosankan, tetapi kami

merasa bahwa perhitungan tersebut layak untuk dilihat karena kegunaan filter Kalman terkait erat dengan sifat matematika distribusi Gaussian.

Model temporal yang kami pertimbangkan menggambarkan jalan acak dari satu variabel status kontinu X_t dengan pengamatan yang bising Z_t . Contohnya adalah indeks "kepercayaan konsumen", yang dapat dimodelkan sebagai perubahan acak berdistribusi Gaussian setiap bulan dan diukur dengan survei konsumen acak yang juga memperkenalkan gangguan pengambilan sampel Gaussian.

Distribusi sebelumnya diasumsikan berdistribusi Gaussian dengan varians σ_0^2 :

$$P(x_0) = \alpha e^{-\frac{1}{2} \left(\frac{x_0 - \mu_0}{\sigma_0^2} \right)^2}$$

(Untuk penyederhanaan, kami menggunakan simbol α yang sama untuk semua konstanta normalisasi di bagian ini.) Model transisi menambahkan gangguan Gaussian dengan varians konstan σ_x^2 ke status saat ini:

$$P(x_{t+1} | x_t) = \alpha e^{-\frac{1}{2} \left(\frac{x_{t+1} - x_t}{\sigma_x^2} \right)^2}$$

Model sensor mengasumsikan noise Gaussian dengan varians σ_z^2 :

$$P(z_t | x_t) = \alpha e^{-\frac{1}{2} \left(\frac{z_t - x_t}{\sigma_z^2} \right)^2}$$

Sekarang, dengan $P(X_0)$ sebelumnya, distribusi prediksi satu langkah berasal dari Persamaan (6.17):

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1 | x_0) P(x_0) dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left(\frac{x_1 - x_0}{\sigma_x^2} \right)^2} e^{-\frac{1}{2} \left(\frac{x_1 - x_0}{\sigma_0^2} \right)^2} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2} \left(\frac{\sigma_0^2 (x_1 - x_0)^2 + \sigma_x^2 (x_1 - x_0)^2}{\sigma_0^2 \sigma_x^2} \right)} dx_0 . \end{aligned}$$

Integral ini terlihat agak rumit. Kunci untuk maju adalah memperhatikan bahwa eksponen adalah jumlah dari dua ekspresi yang kuadrat di x_0 dan karenanya merupakan kuadrat di x_0 . Trik sederhana yang dikenal sebagai melengkapi kuadrat memungkinkan penulisan ulang sembarang kuadrat $ax_0^2 + bx_0 + c$ sebagai jumlah suku kuadrat $a(x_0 - \frac{-b}{2a})^2$ dan suku residual $c - \frac{b^2}{4a}$ yang independen dari x_0 . Suku residual dapat diambil di luar integral, sehingga kita memperoleh:

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(c - \frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a\left(x_0 - \frac{b}{2a}\right)^2\right)} dx_0$$

Sekarang integral tersebut hanyalah integral Gaussian atas rentang penuhnya, yang hanya 1. Jadi, kita hanya memiliki suku residual dari kuadrat. Kemudian, kita perhatikan bahwa suku residual adalah kuadrat di x_1 ; pada kenyataannya, setelah penyederhanaan, kita memperoleh:

$$P(x_1) = \alpha e^{-\frac{1}{2}\left(\frac{(x_1 - \mu_0)^2}{o_0^2 + o_x^2}\right)}$$

Artinya, distribusi prediksi satu langkah adalah Gaussian dengan mean μ_0 yang sama dan varians yang sama dengan jumlah varians asli o_0^2 dan varians transisi o_x^2 .

Untuk melengkapi langkah pembaruan, kita perlu mengkondisikan observasi pada langkah waktu pertama, yaitu, z_1 . Dari Persamaan (6.18), ini diberikan oleh:

$$\begin{aligned} P(x_1 | z_1) &= \alpha P(z_1 | x_1) P(x_1) \\ &= \alpha e^{-\frac{1}{2}\left(\frac{(z_1 - x_1)^2}{o_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1 - \mu_0)^2}{o_0^2 + o_x^2}\right)} \end{aligned}$$

Sekali lagi, kita gabungkan eksponen dan lengkapi kuadratnya, sehingga diperoleh:

$$P(x_1 | z_1) = \alpha e^{-\frac{1}{2}\left(\frac{\left(x_1 - \frac{(o_0^2 + o_x^2)z_1 + o_z^2\mu_0}{o_0^2 + o_x^2 + o_z^2}\right)^2}{\frac{(o_0^2 + o_x^2)o_z^2}{(o_0^2 + o_x^2) + o_z^2}}\right)} \quad (6.19)$$

Dengan demikian, setelah satu siklus pembaruan, kita memiliki distribusi Gaussian baru untuk variabel status.

Dari rumus Gaussian dalam Persamaan (6.19), kita melihat bahwa rata-rata dan simpangan baku yang baru dapat dihitung dari rata-rata dan simpangan baku yang lama sebagai berikut:

$$\mu_{t+1} = \frac{(o_t^2 + o_x^2)z_{t+1} + o_z^2\mu_t}{o_t^2 + o_x^2 + o_z^2} \quad \text{dan} \quad o_{t+1}^2 = \frac{(o_t^2 + o_x^2)o_z^2}{o_t^2 + o_x^2 + o_z^2}. \quad (6.20)$$

Gambar 6.10 menunjukkan satu siklus pembaruan untuk nilai-nilai tertentu dari model transisi dan sensor. Persamaan (6.20) memainkan peran yang sama persis dengan persamaan penyaringan umum (15.5) atau persamaan penyaringan HMM (6.12). Namun, karena sifat khusus dari distribusi Gaussian, persamaan tersebut memiliki beberapa sifat tambahan yang menarik. Pertama, kita dapat menginterpretasikan perhitungan untuk rata-rata baru μ_{t+1} hanya sebagai rata-rata tertimbang dari pengamatan baru z_{t+1} dan rata-rata lama μ_t .

Jika pengamatan tidak dapat diandalkan, maka σ_z^2 besar dan kita lebih memperhatikan rata-rata lama; jika rata-rata lama tidak dapat diandalkan (σ_t^2 besar) atau prosesnya sangat tidak dapat diprediksi (σ_x^2 besar), maka kita lebih memperhatikan pengamatan. Kedua, perhatikan bahwa pembaruan untuk varians σ_{t+1}^2 tidak bergantung pada pengamatan. Oleh karena itu, kita dapat menghitung terlebih dahulu seperti apa urutan nilai variansnya. Ketiga, urutan nilai varians konvergen dengan cepat ke nilai tetap yang hanya bergantung pada σ_x^2 dan σ_z^2 , sehingga menyederhanakan perhitungan selanjutnya secara substansial.

Kasus Umum

Derivasi sebelumnya menggambarkan sifat utama distribusi Gaussian yang memungkinkan penyaringan Kalman berfungsi: fakta bahwa eksponen berbentuk kuadrat. Hal ini berlaku tidak hanya untuk kasus univariat; distribusi Gaussian multivariat penuh memiliki bentuk:

$$N(\mu, \Sigma)(x) = \alpha e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}.$$

Mengalikan suku-suku dalam eksponen memperjelas bahwa eksponen juga merupakan fungsi kuadrat dari nilai x_t dalam x . Seperti dalam kasus univariat, pembaruan penyaringan mempertahankan sifat Gaussian dari distribusi status.

Pertama-tama mari kita definisikan model temporal umum yang digunakan dengan penyaringan Kalman. Baik model transisi maupun model sensor memungkinkan transformasi linier dengan derau Gaussian aditif. Jadi, kita memiliki:

$$\begin{aligned} P(x_{t+1} | x_t) &= N(Fx_t, \Sigma_x)(x_{t+1}) \\ P(z_t | x_t) &= N(Hx_t, \Sigma_z)(z_t), \end{aligned} \quad (6.21)$$

di mana F dan Σ_x adalah matriks yang menggambarkan model transisi linier dan kovariansi kebisingan transisi, dan H dan Σ_z adalah matriks yang sesuai untuk model sensor. Sekarang persamaan pembaruan untuk mean dan kovariansi, dalam bentuk yang mengerikan dan mengerikan, adalah:

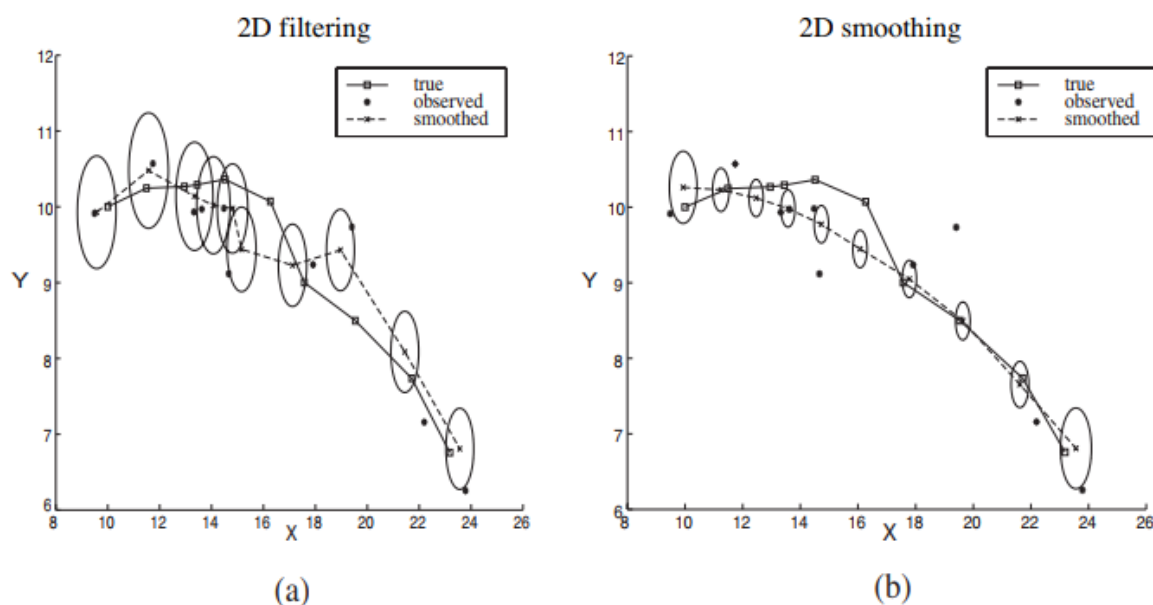
$$\begin{aligned} \mu_{t+1} &= F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \\ \Sigma_{t+1} &= I - K_{t+1}H(F\Sigma_t F^\top + \Sigma_x), \end{aligned} \quad (6.22)$$

di mana $K_{t+1} = (F\Sigma_t F^\top + \Sigma_x)H^\top (H(F\Sigma_t F^\top + \Sigma_x)H^\top + \Sigma_z)^{-1}$ disebut matriks penguatan Kalman. Percaya atau tidak, persamaan ini memiliki beberapa pengertian intuitif. Misalnya, perhatikan pembaruan untuk estimasi keadaan rata-rata μ . Suku $F\mu_t$ adalah keadaan yang diprediksi pada $t + 1$, jadi $HF\mu_t$ adalah observasi yang diprediksi. Oleh karena itu, suku $z_{t+1} - HF\mu_t$ menunjukkan kesalahan dalam observasi yang diprediksi. Ini dikalikan dengan K_{t+1} untuk mengoreksi keadaan yang diprediksi; oleh karena itu, K_{t+1} adalah ukuran seberapa serius

untuk menanggapi observasi baru relatif terhadap prediksi. Seperti dalam Persamaan (6.20), kita juga memiliki sifat bahwa pembaruan varians tidak bergantung pada observasi.

Oleh karena itu, urutan nilai untuk Σ_t dan K_t dapat dihitung secara luring, dan perhitungan aktual yang diperlukan selama pelacakan daring cukup sederhana. Untuk mengilustrasikan persamaan ini saat bekerja, kami telah menerapkannya pada masalah pelacakan objek yang bergerak pada bidang $X - Y$. Variabel keadaan adalah $\mathbf{x} = (X, Y, \dot{X}, \dot{Y})^T$, jadi F, Σ_x, H , dan Σ_z adalah matriks 4×4 . Gambar 6.11(a) menunjukkan lintasan sebenarnya, serangkaian pengamatan yang bising, dan lintasan yang diestimasi oleh penyaringan Kalman, bersama dengan kovariansi yang ditunjukkan oleh kontur satu standar deviasi. Proses penyaringan berfungsi dengan baik dalam melacak gerakan aktual, dan, seperti yang diharapkan, varians dengan cepat mencapai titik tetap.

Kita juga dapat memperoleh persamaan untuk penghalusan serta penyaringan dengan model Gaussian linier. Hasil penghalusan ditunjukkan pada Gambar 6.11(b). Perhatikan bagaimana varians dalam estimasi posisi berkurang tajam, kecuali di ujung lintasan (mengapa?), dan bahwa lintasan yang diestimasi jauh lebih halus.



Gambar 6.11 (a) Hasil penyaringan Kalman untuk objek yang bergerak pada bidang $X - Y$, yang menunjukkan lintasan sebenarnya (dari kiri ke kanan), serangkaian pengamatan yang terganggu, dan lintasan yang diperkirakan dengan penyaringan Kalman. Varians dalam perkiraan posisi ditunjukkan oleh oval. (b) Hasil penghalusan Kalman untuk urutan pengamatan yang sama.

Penerapan Penyaringan Kalman

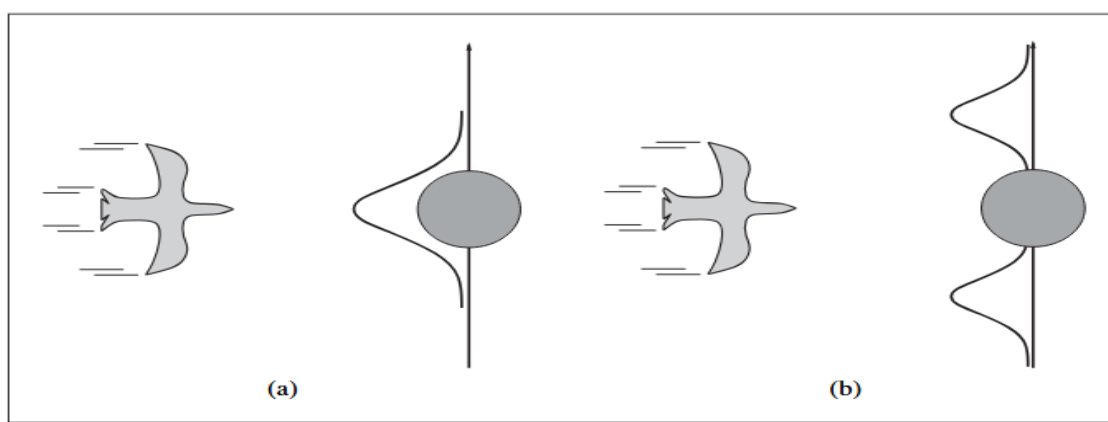
Filter Kalman dan elaborasinya digunakan dalam berbagai macam aplikasi. Aplikasi "klasik" adalah dalam pelacakan radar pesawat terbang dan rudal. Aplikasi terkait termasuk pelacakan akustik kapal selam dan kendaraan darat serta pelacakan visual kendaraan dan orang. Dalam konteks yang sedikit lebih esoteris, filter Kalman digunakan untuk

merekonstruksi lintasan partikel dari foto ruang gelembung dan arus laut dari pengukuran permukaan satelit. Jangkauan aplikasinya jauh lebih luas daripada sekadar pelacakan gerakan: sistem apa pun yang dicirikan oleh variabel status kontinu dan pengukuran bising dapat digunakan. Sistem tersebut termasuk pabrik pulp, pabrik kimia, reaktor nuklir, ekosistem tanaman, dan ekonomi nasional.

Fakta bahwa penyaringan Kalman dapat diterapkan pada suatu sistem tidak berarti bahwa hasilnya akan valid atau berguna. Asumsi yang dibuat—transisi Gaussian linear dan model sensor—sangat kuat. Penyaring Kalman yang diperluas (EKF) berupaya mengatasi nonlinieritas dalam sistem yang dimodelkan. Suatu sistem bersifat nonlinier jika model transisi tidak dapat dijelaskan sebagai perkalian matriks dari vektor status, seperti dalam Persamaan (6.21). EKF bekerja dengan memodelkan sistem sebagai linier lokal dalam x_t di wilayah $x_t = \mu_t$, rata-rata distribusi status saat ini. Ini bekerja dengan baik untuk sistem yang halus dan berperilaku baik dan memungkinkan pelacak untuk mempertahankan dan memperbarui distribusi status Gaussian yang merupakan perkiraan yang wajar terhadap posterior yang sebenarnya.

Apa artinya bagi suatu sistem untuk menjadi "tidak halus" atau "berperilaku buruk"? Secara teknis, ini berarti bahwa terdapat nonlinieritas signifikan dalam respons sistem dalam wilayah yang "dekat" (menurut kovariansi Σ_t) dengan rata-rata μ saat ini. Untuk memahami gagasan ini dalam istilah nonteknis, perhatikan contoh upaya melacak burung saat terbang melalui hutan.

Burung itu tampak sedang menuju dengan kecepatan tinggi langsung ke batang pohon. Filter Kalman, baik reguler maupun yang diperluas, hanya dapat membuat prediksi Gaussian tentang lokasi burung, dan rata-rata Gaussian ini akan berpusat pada batang pohon, seperti yang ditunjukkan pada Gambar 6.12(a). Sebaliknya, model burung yang wajar akan memprediksi tindakan mengelak ke satu sisi atau sisi lain, seperti yang ditunjukkan pada Gambar 6.12(b). Model seperti itu sangat nonlinier, karena keputusan burung sangat bervariasi tergantung pada lokasi tepatnya yang relatif terhadap batang pohon.



Gambar 6.12 Seekor burung terbang ke arah pohon (tampilan atas). (a) Filter Kalman akan memprediksi lokasi burung menggunakan satu Gaussian yang berpusat pada rintangan. (b)

Model yang lebih realistis memperhitungkan tindakan menghindari burung, memprediksi bahwa burung akan terbang ke satu sisi atau sisi lainnya.

Untuk menangani contoh-contoh seperti ini, kita jelas membutuhkan bahasa yang lebih ekspresif untuk merepresentasikan perilaku sistem yang sedang dimodelkan. Dalam komunitas teori kontrol, yang mana masalah seperti manuver mengelak oleh pesawat menimbulkan kesulitan yang sama, solusi standarnya adalah filter Kalman switching. Dalam pendekatan ini, beberapa filter Kalman berjalan secara paralel, masing-masing menggunakan model sistem yang berbeda—misalnya, satu untuk penerbangan lurus, satu untuk belokan tajam ke kiri, dan satu untuk belokan tajam ke kanan. Jumlah prediksi yang dibobot digunakan, di mana bobotnya bergantung pada seberapa baik setiap filter sesuai dengan data saat ini.

Kita akan melihat di bagian berikutnya bahwa ini hanyalah kasus khusus dari model jaringan Bayesian dinamis umum, yang diperoleh dengan menambahkan variabel status "manuver" diskrit ke jaringan yang ditunjukkan pada Gambar 6.9.

6.5 JARINGAN BAYESIAN DINAMIS

Jaringan Bayesian dinamis, atau DBN, adalah jaringan Bayesian yang merepresentasikan model probabilitas temporal. Kita telah melihat contoh DBN: jaringan payung pada Gambar 6.2 dan jaringan filter Kalman pada Gambar 6.9. Secara umum, setiap irisan DBN dapat memiliki sejumlah variabel status X_t dan variabel bukti E_t . Untuk menyederhanakannya, kita berasumsi bahwa variabel dan tautannya direplikasi secara tepat dari irisan ke irisan dan bahwa DBN merepresentasikan proses Markov orde pertama, sehingga setiap variabel hanya dapat memiliki induk di irisannya sendiri atau irisan sebelumnya.

Harus jelas bahwa setiap model Markov tersembunyi dapat direpresentasikan sebagai DBN dengan satu variabel status dan satu variabel bukti. Setiap DBN variabel diskrit juga dapat direpresentasikan sebagai HMM, kita dapat menggabungkan semua variabel status di DBN menjadi satu variabel status yang nilainya adalah semua tupel nilai yang mungkin dari variabel status individual. Sekarang, jika setiap HMM adalah DBN dan setiap DBN dapat diterjemahkan ke dalam HMM, apa bedanya? Perbedaannya adalah, dengan menguraikan status sistem yang kompleks menjadi variabel penyusunnya, dapat memanfaatkan kelangkaan dalam model probabilitas temporal.

Misalkan, misalnya, bahwa DBN memiliki 20 variabel status Boolean, yang masing-masing memiliki tiga induk di irisan sebelumnya. Maka model transisi DBN memiliki $20 \times 2^3 = 160$ probabilitas, sedangkan HMM yang sesuai memiliki 2^{20} status dan oleh karena itu 2^{40} , atau kira-kira satu triliun, probabilitas dalam matriks transisi. Ini buruk setidaknya karena tiga alasan: pertama, HMM itu sendiri membutuhkan lebih banyak ruang; kedua, matriks transisi yang besar membuat inferensi HMM jauh lebih mahal; dan ketiga, masalah mempelajari sejumlah besar parameter membuat model HMM murni tidak cocok untuk masalah besar. Hubungan antara DBN dan HMM secara kasar analog dengan hubungan antara jaringan Bayesian biasa dan distribusi gabungan tabulasi penuh.

Kami telah menjelaskan bahwa setiap model filter Kalman dapat direpresentasikan dalam DBN dengan variabel kontinu dan distribusi kondisional Gaussian linier (Gambar 6.9). Seharusnya jelas dari pembahasan di akhir bagian sebelumnya bahwa tidak setiap DBN dapat direpresentasikan oleh model filter Kalman. Dalam filter Kalman, distribusi status saat ini selalu merupakan distribusi Gaussian multivariat tunggal—yaitu, satu "benjolan" di lokasi tertentu. DBN, di sisi lain, dapat memodelkan distribusi arbitrer. Untuk banyak aplikasi dunia nyata, fleksibilitas ini penting. Pertimbangkan, misalnya, lokasi kunci saya saat ini. Kunci itu mungkin ada di saku saya, di meja samping tempat tidur, di meja dapur, tergantung di pintu depan, atau terkunci di dalam mobil. Satu tonjolan Gaussian yang mencakup semua tempat ini harus mengalokasikan probabilitas signifikan ke kunci yang berada di udara di aula depan. Aspek dunia nyata seperti agen yang bertujuan, rintangan, dan kantong memperkenalkan "nonlinieritas" yang memerlukan kombinasi variabel diskrit dan kontinu untuk mendapatkan model yang masuk akal.

Membangun DBN

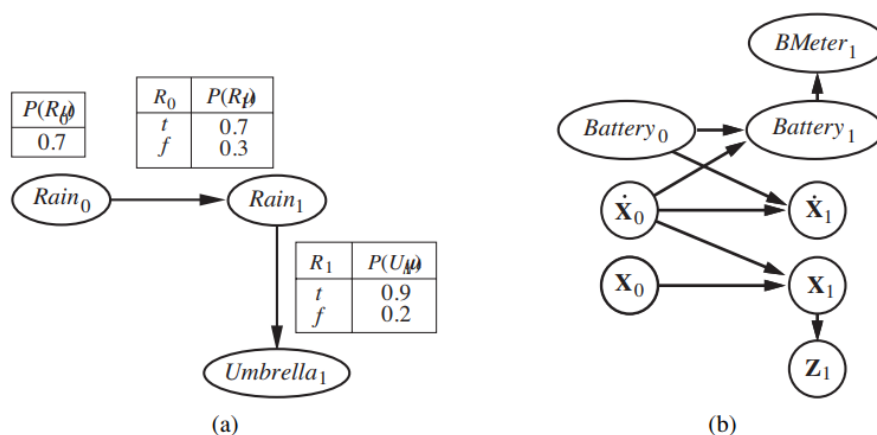
Untuk membangun DBN, seseorang harus menentukan tiga jenis informasi: distribusi sebelumnya atas variabel status, $P(X_0)$; model transisi $P(X_{t+1} | X_t)$; dan model sensor $P(E_t | X_t)$. Untuk menentukan model transisi dan sensor, seseorang juga harus menentukan topologi koneksi antara irisan yang berurutan dan antara variabel status dan bukti. Karena model transisi dan sensor diasumsikan stasioner—sama untuk semua t —maka paling mudah untuk menentukannya untuk irisan pertama. Misalnya, spesifikasi DBN lengkap untuk dunia payung diberikan oleh jaringan tiga simpul yang ditunjukkan pada Gambar 6.13(a). Dari spesifikasi ini, DBN lengkap dengan jumlah irisan waktu yang tidak terbatas dapat dibangun sesuai kebutuhan dengan menyalin irisan pertama.

Sekarang mari kita pertimbangkan contoh yang lebih menarik: memantau robot bertenaga baterai yang bergerak di bidang $X - Y$, seperti yang diperkenalkan di akhir Bagian 6.1. Pertama, kita perlu variabel status, yang akan mencakup $X_t = (X_t, Y_t)$ untuk posisi dan $\dot{X}_t = (\dot{X}_t, \dot{Y}_t)$ untuk kecepatan.

Kita asumsikan beberapa metode pengukuran posisi—mungkin kamera tetap atau GPS (*Global Positioning System*) di dalam pesawat—yang menghasilkan pengukuran Z_t . Posisi pada langkah waktu berikutnya bergantung pada posisi dan kecepatan saat ini, seperti pada model filter Kalman standar. Kecepatan pada langkah berikutnya bergantung pada kecepatan saat ini dan status baterai. Kita tambahkan $Battery_t$ untuk mewakili level pengisian daya baterai aktual, yang memiliki induk level baterai sebelumnya dan kecepatan, dan kita tambahkan $BMeter_t$, yang mengukur level pengisian daya baterai. Ini memberi kita model dasar yang ditunjukkan pada Gambar 6.13(b).

Ada baiknya melihat lebih dalam pada sifat model sensor untuk $BMeter_t$. Mari kita asumsikan, untuk kesederhanaan, bahwa $Battery_t$ dan $BMeter_t$ dapat mengambil nilai diskrit 0 hingga 5. Jika meter selalu akurat, maka CPT $P(BMeter_t | Battery_t)$ harus memiliki probabilitas 1.0 "sepanjang diagonal" dan probabilitas 0.0 di tempat lain. Pada kenyataannya, gangguan selalu merayap ke dalam pengukuran. Untuk pengukuran berkelanjutan, distribusi Gaussian dengan varians kecil dapat digunakan. Untuk variabel diskrit kita, kita dapat

memperkirakan Gaussian menggunakan distribusi di mana probabilitas kesalahan menurun dengan cara yang tepat, sehingga probabilitas kesalahan besar sangat kecil. Kami menggunakan istilah model kesalahan Gaussian untuk mencakup versi berkelanjutan dan diskrit.



Gambar 6.13 (a) Spesifikasi model sebelumnya, model transisi, dan model sensor untuk DBN payung. Semua irisan berikutnya diasumsikan sebagai salinan irisan 1. (b) DBN sederhana untuk gerakan robot di bidang $X - Y$.

Siapa pun yang memiliki pengalaman langsung dengan robotika, kontrol proses terkomputerisasi, atau bentuk penginderaan otomatis lainnya akan dengan mudah bersaksi atas fakta bahwa sejumlah kecil gangguan pengukuran sering kali merupakan masalah yang paling kecil. Sensor yang sebenarnya gagal. Ketika sensor gagal, ia tidak selalu mengirimkan sinyal yang mengatakan, "Oh, omong-omong, data yang akan saya kirimkan kepada Anda adalah omong kosong." Sebaliknya, ia hanya mengirimkan omong kosong. Jenis kegagalan yang paling sederhana disebut kegagalan sementara, di mana sensor terkadang memutuskan untuk mengirimkan sesuatu yang tidak masuk akal. Misalnya, sensor level baterai mungkin memiliki kebiasaan mengirimkan angka nol ketika seseorang menabrak robot, bahkan jika baterai terisi penuh.

Mari kita lihat apa yang terjadi ketika kegagalan sementara terjadi dengan model kesalahan Gaussian yang tidak mengakomodasi kegagalan tersebut. Misalnya, anggaplah robot tersebut duduk dengan tenang dan mengamati 20 pembacaan baterai berturut-turut sebesar 5. Kemudian meteran baterai mengalami kejang sementara dan pembacaan berikutnya adalah $BMeter_{21} = 0$. Apa yang akan membuat kita percaya dengan model kesalahan Gaussian sederhana tentang Baterai 21? Menurut aturan Bayes, jawabannya bergantung pada model sensor $P(BMeter_{21} = 0 | Battery_{21})$ dan prediksi $P(Battery_{21} | BMeter_{1:20})$.

Jika probabilitas kesalahan sensor yang besar secara signifikan lebih kecil kemungkinannya daripada probabilitas transisi ke $Battery_{21} = 0$, bahkan jika yang terakhir sangat tidak mungkin, maka distribusi posterior akan menetapkan probabilitas tinggi pada baterai yang kosong. Pembacaan kedua sebesar 0 pada $t = 22$ akan membuat kesimpulan ini hampir pasti. Jika kegagalan sementara kemudian menghilang dan pembacaan kembali ke 5

dari $t = 23$ dan seterusnya, perkiraan untuk level baterai akan dengan cepat kembali ke 5, seolah-olah dengan sihir. Rangkaian peristiwa ini diilustrasikan dalam kurva atas Gambar 6.14(a), yang menunjukkan nilai yang diharapkan dari $Battery_t$ dari waktu ke waktu, menggunakan model galat Gaussian diskrit.

Meskipun terjadi pemulihan, ada saat ($t = 22$) ketika robot yakin bahwa baterainya kosong; mungkin, robot akan mengirimkan sinyal mayday dan mati. Sayangnya, model sensornya yang terlalu sederhana telah membuatnya tersesat. Bagaimana ini bisa diperbaiki? Pertimbangkan contoh yang sudah dikenal dari pengendaraan manusia sehari-hari: di tikungan tajam atau bukit curam, lampu peringatan "tangki bahan bakar kosong" terkadang menyala. Daripada mencari telepon darurat, seseorang cukup mengingat bahwa pengukur bahan bakar terkadang memberikan galat yang sangat besar saat bahan bakar terciprat di dalam tangki. Moral dari cerita ini adalah sebagai berikut: agar sistem dapat menangani kegagalan sensor dengan benar, model sensor harus mencakup kemungkinan kegagalan.

Jenis model kegagalan paling sederhana untuk sensor memungkinkan probabilitas tertentu bahwa sensor akan mengembalikan beberapa nilai yang sepenuhnya salah, terlepas dari keadaan dunia yang sebenarnya. Misalnya, jika meteran baterai gagal dengan mengembalikan 0, kita dapat mengatakan bahwa:

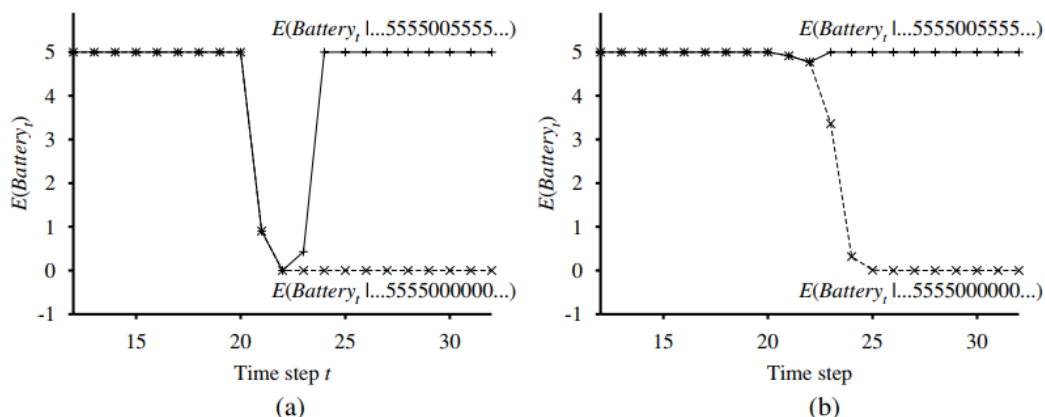
$$P(BMeter_t = 0 \mid Battery_t = 5) = 0.03 ,$$

yang mungkin jauh lebih besar daripada probabilitas yang ditetapkan oleh model galat Gaussian sederhana. Mari kita sebut ini model kegagalan sementara. Bagaimana ini membantu ketika kita dihadapkan pada pembacaan 0? Asalkan probabilitas baterai kosong yang diprediksi, menurut pembacaan sejauh ini, jauh lebih kecil dari 0,03, maka penjelasan terbaik dari pengamatan $BMeter_{21} = 0$ adalah bahwa sensor telah gagal sementara. Secara intuitif, kita dapat menganggap keyakinan tentang level baterai memiliki sejumlah "kelembaman" yang membantu mengatasi gangguan sementara dalam pembacaan meter. Kurva atas pada Gambar 6.14(b) menunjukkan bahwa model kegagalan sementara dapat menangani kegagalan sementara tanpa perubahan besar dalam keyakinan.

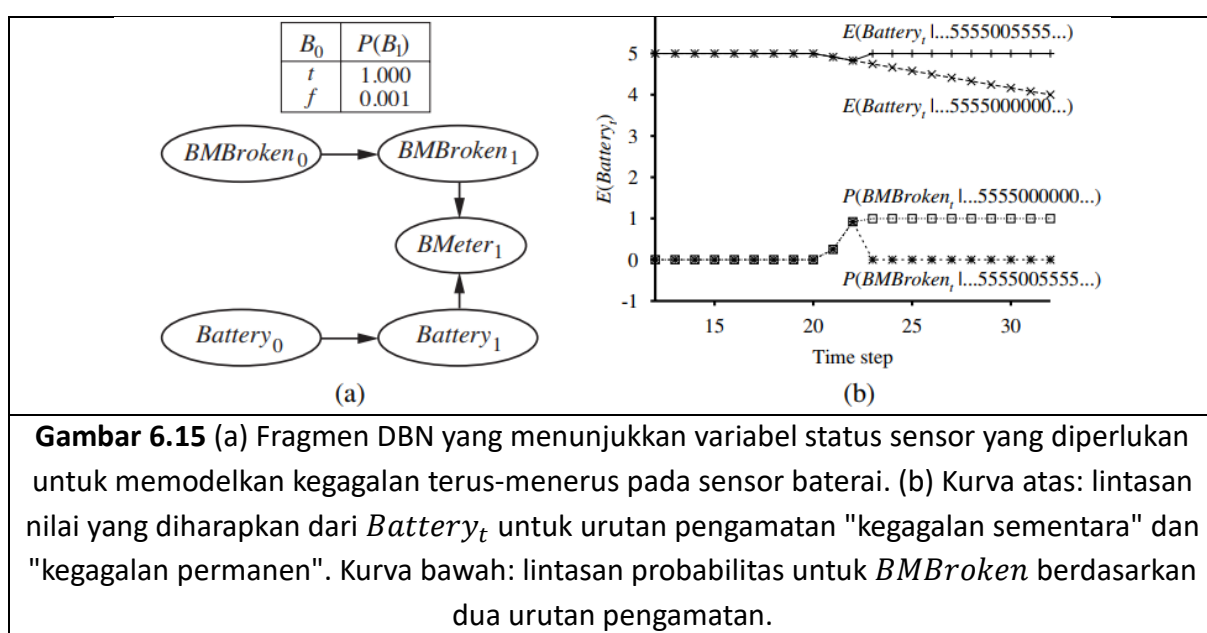
Sekian tentang gangguan sementara. Bagaimana dengan kegagalan sensor yang terus-menerus? Sayangnya, kegagalan semacam ini terlalu umum. Jika sensor mengembalikan 20 pembacaan 5 diikuti oleh 20 pembacaan 0, maka model kegagalan sensor sementara yang dijelaskan dalam paragraf sebelumnya akan mengakibatkan robot secara bertahap mulai percaya bahwa baterainya kosong padahal sebenarnya meteran tersebut mungkin telah gagal. Kurva bawah pada Gambar 6.14(b) menunjukkan "lintasan" keyakinan untuk kasus ini. Dengan $t = 25$ —lima pembacaan 0—robot yakin bahwa baterainya kosong. Jelas, kami lebih suka robot percaya bahwa meteran baterainya rusak—jika memang ini adalah peristiwa yang lebih mungkin terjadi.

Tidak mengherankan, untuk menangani kegagalan terus-menerus, kami memerlukan model kegagalan terus-menerus yang menggambarkan bagaimana sensor berperilaku dalam kondisi normal dan setelah kegagalan. Untuk melakukan ini, kami perlu menambah status

sistem dengan variabel tambahan, misalnya, $BMBroken$, yang menggambarkan status meteran baterai. Kegigihan kegagalan harus dimodelkan oleh busur yang menghubungkan $BMBroken_0$ ke $BMBroken_1$. Busur persistensi ini memiliki CPT yang memberikan kemungkinan kecil kegagalan dalam setiap langkah waktu tertentu, misalnya, 0.001, tetapi menetapkan bahwa sensor tetap rusak setelah rusak. Ketika sensor dalam keadaan baik, model sensor untuk $BMeter$ identik dengan model kegagalan sementara; ketika sensor rusak, dikatakan bahwa $BMeter$ selalu 0, terlepas dari daya baterai yang sebenarnya.



Gambar 6.14 (a) Kurva atas: lintasan nilai yang diharapkan dari $Battery_t$ untuk urutan pengamatan yang terdiri dari semua angka 5 kecuali 0 pada $t = 21$ dan $t = 22$, menggunakan model galat Gaussian sederhana. Kurva bawah: lintasan saat pengamatan tetap pada angka 0 dari $t = 21$ dan seterusnya. (b) Eksperimen yang sama dijalankan dengan model kegagalan sementara. Perhatikan bahwa kegagalan sementara ditangani dengan baik, tetapi kegagalan yang terus-menerus mengakibatkan pesimisme yang berlebihan tentang pengisian daya baterai.



Gambar 6.15 (a) Fragmen DBN yang menunjukkan variabel status sensor yang diperlukan untuk memodelkan kegagalan terus-menerus pada sensor baterai. (b) Kurva atas: lintasan nilai yang diharapkan dari $Battery_t$ untuk urutan pengamatan "kegagalan sementara" dan "kegagalan permanen". Kurva bawah: lintasan probabilitas untuk $BMBroken$ berdasarkan dua urutan pengamatan.

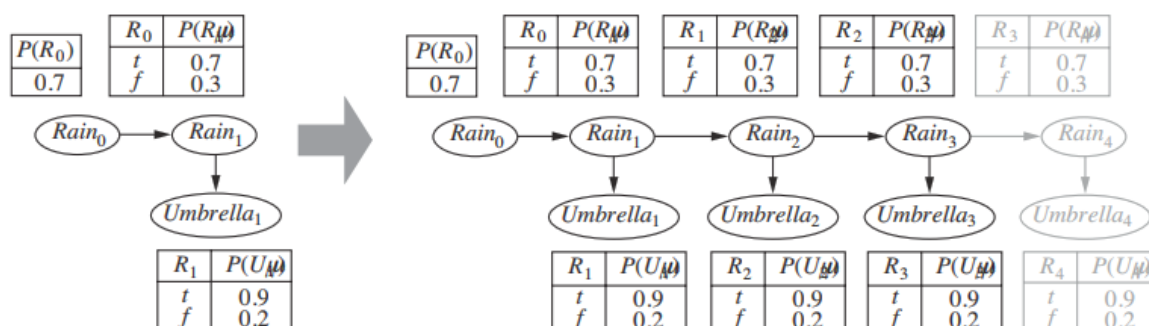
Model kegagalan berkelanjutan untuk sensor baterai ditunjukkan pada Gambar 6.15(a). Kinerjanya pada dua urutan data (gangguan sementara dan kegagalan berkelanjutan) ditunjukkan pada Gambar 6.15(b). Ada beberapa hal yang perlu diperhatikan tentang kurva ini. Pertama, dalam kasus gangguan sementara, kemungkinan sensor rusak meningkat secara signifikan setelah pembacaan 0 kedua, tetapi segera turun kembali ke nol setelah 5 diamati. Kedua, dalam kasus kegagalan berkelanjutan, kemungkinan sensor rusak meningkat dengan cepat hingga hampir 1 dan tetap di sana. Akhirnya, setelah sensor diketahui rusak, robot hanya dapat berasumsi bahwa baterainya habis pada tingkat "normal", seperti yang ditunjukkan oleh tingkat $E(Battery_t | \dots)$ yang menurun secara bertahap.

Sejauh ini, kita baru menyentuh permukaan masalah representasi proses yang kompleks. Variasi model transisi sangat besar, mencakup topik yang berbeda seperti pemodelan sistem endokrin manusia dan pemodelan beberapa kendaraan yang melaju di jalan bebas hambatan. Pemodelan sensor juga merupakan subbidang yang luas, tetapi bahkan fenomena yang tidak kentara, seperti pergeseran sensor, dekalibrasi tiba-tiba, dan efek kondisi eksogen (seperti cuaca) pada pembacaan sensor, dapat ditangani dengan representasi eksplisit dalam jaringan Bayesian yang dinamis.

Inferensi Eksak dalam DBN

Setelah membuat sketsa beberapa ide untuk merepresentasikan proses kompleks sebagai DBN, sekarang kita beralih ke pertanyaan tentang inferensi. Dalam arti tertentu, pertanyaan ini telah terjawab: jaringan Bayesian yang dinamis adalah jaringan Bayesian, dan kita telah memiliki algoritme untuk inferensi dalam jaringan Bayesian. Dengan urutan pengamatan, seseorang dapat membangun representasi jaringan Bayesian penuh dari DBN dengan mereplikasi irisan hingga jaringan cukup besar untuk mengakomodasi pengamatan, seperti pada Gambar 6.16.

Teknik ini dalam konteks model probabilitas relasional, disebut unrolling. (Secara teknis, DBN setara dengan jaringan semi-tak terbatas yang diperoleh dengan melakukan unrolling selamanya. Irisan yang ditambahkan setelah pengamatan terakhir tidak memiliki efek pada inferensi dalam periode pengamatan dan dapat dihilangkan.) Setelah DBN di-unrolling, seseorang dapat menggunakan salah satu algoritme inferensi—penghapusan variabel, metode pengelompokan, dan sebagainya.



Gambar 6.16 Membuka jaringan Bayesian yang dinamis: irisan direplikasi untuk mengakomodasi urutan observasi $Umbrella_{1:3}$. Irisan selanjutnya tidak memiliki efek pada inferensi dalam periode observasi.

Sayangnya, penerapan unrolling yang naif tidak akan terlalu efisien. Jika kita ingin melakukan penyaringan atau penghalusan dengan urutan pengamatan yang panjang $e_{1:t}$, jaringan yang tidak di-unroll akan memerlukan ruang $O(t)$ dan dengan demikian akan tumbuh tanpa batas saat lebih banyak pengamatan ditambahkan. Selain itu, jika kita cukup menjalankan algoritme inferensi baru setiap kali pengamatan ditambahkan, waktu inferensi per pembaruan juga akan meningkat sebagai $O(t)$.

Kita melihat bahwa waktu dan ruang konstan per pembaruan penyaringan dapat dicapai jika perhitungan dapat dilakukan secara rekursif. Pada dasarnya, pembaruan penyaringan dalam Persamaan (6.5) bekerja dengan menjumlahkan variabel status dari langkah waktu sebelumnya untuk mendapatkan distribusi untuk langkah waktu yang baru.

Menjumlahkan variabel adalah persis apa yang dilakukan algoritma eliminasi variabel (Gambar 5.11), dan ternyata menjalankan eliminasi variabel dengan variabel dalam urutan temporal persis meniru operasi pembaruan penyaringan rekursif dalam Persamaan (6.5). Algoritme yang dimodifikasi menyimpan paling banyak dua irisan dalam memori pada satu waktu: dimulai dengan irisan 0, kita tambahkan irisan 1, lalu jumlahkan irisan 0, lalu tambahkan irisan 2, lalu jumlahkan irisan 1, dan seterusnya. Dengan cara ini, kita dapat mencapai ruang dan waktu yang konstan per pembaruan penyaringan. (Kinerja yang sama dapat dicapai dengan modifikasi yang sesuai pada algoritma pengelompokan.)

Sekian kabar baiknya; sekarang untuk berita buruknya: Ternyata "konstanta" untuk kompleksitas waktu dan ruang per pembaruan, dalam hampir semua kasus, eksponensial dalam jumlah variabel status. Yang terjadi adalah, saat eliminasi variabel berlanjut, faktor-faktornya bertambah hingga mencakup semua variabel status (atau, lebih tepatnya, semua variabel status yang memiliki induk dalam irisan waktu sebelumnya). Ukuran faktor maksimum adalah $O(d^{n+k})$ dan total biaya pembaruan per langkah adalah $O(nd^{n+k})$, di mana d adalah ukuran domain variabel dan k adalah jumlah maksimum induk dari setiap variabel status.

Tentu saja, ini jauh lebih sedikit daripada biaya pembaruan HMM, yang adalah $O(d^{2n})$, tetapi masih tidak layak untuk sejumlah besar variabel. Fakta suram ini agak sulit diterima. Artinya adalah bahwa meskipun kita dapat menggunakan DBN untuk mewakili proses temporal yang sangat kompleks dengan banyak variabel yang terhubung jarang, kita tidak dapat bernalar secara efisien dan tepat tentang proses tersebut.

Model DBN itu sendiri, yang mewakili distribusi gabungan sebelumnya atas semua variabel, dapat difaktorkan ke dalam CPT penyusunnya, tetapi distribusi gabungan posterior yang dikondisikan pada urutan observasi—yaitu, pesan maju—umumnya tidak dapat difaktorkan. Sejauh ini, belum ada yang menemukan cara untuk mengatasi masalah ini, meskipun faktanya banyak bidang sains dan teknik penting akan mendapat manfaat besar dari solusinya. Jadi, kita harus kembali ke metode perkiraan.

```

function PARTICLE-FILTERING( $e, N, dbn$ ) returns a set of samples for the next time
step
  inputs:  $e$ , the new incoming evidence
             $N$ , the number of samples to be maintained
             $dbn$ , a DBN with prior  $P(X_0)$ , transition model  $P(X_1|X_0)$ , sensormodel
 $P(E_1|X_1)$ 
  persistent:  $S$ , a vector of samples of size  $N$ , initially generated from  $P(X_0)$ 
  local variables:  $W$ , a vector of weights of size  $N$ 

  for  $i = 1$  to  $N$  do
     $S[i] \leftarrow$  sample from  $P(X_1 | X_0 = S[i])$  /* step 1 */

```

Gambar 6.17 Algoritma penyaringan partikel diimplementasikan sebagai operasi pembaruan rekursif dengan status (himpunan sampel). Setiap operasi pengambilan sampel melibatkan pengambilan sampel variabel irisan yang relevan dalam urutan topologi, seperti dalam PRIOR-SAMPLE. Operasi WEIGHTED-SAMPLE-WITH-REPLACEMENT dapat diimplementasikan untuk berjalan dalam waktu yang diharapkan $O(N)$. Nomor langkah merujuk pada deskripsi dalam teks.

Inferensi Perkiraan dalam DBN

Bagian sebelumnya menjelaskan dua algoritme perkiraan: pembobotan kemungkinan (Gambar 5.15) dan *Markov chain Monte Carlo*. Dari keduanya, yang pertama paling mudah diadaptasi ke dalam konteks DBN. (Algoritme penyaringan MCMC dijelaskan secara singkat dalam catatan di akhir bab.)

Akan tetapi, kita akan melihat bahwa beberapa perbaikan diperlukan atas algoritme pembobotan kemungkinan standar sebelum metode praktis muncul. Ingat bahwa pembobotan kemungkinan bekerja dengan mengambil sampel simpul non-bukti dari jaringan dalam urutan topologi, memberi bobot pada setiap sampel berdasarkan kemungkinan yang diberikannya pada variabel bukti yang diamati. Seperti halnya algoritme eksak, kita dapat menerapkan pembobotan kemungkinan secara langsung ke DBN yang belum digulirkan, tetapi ini akan mengalami masalah yang sama dengan meningkatnya persyaratan waktu dan ruang per pembaruan seiring dengan bertambahnya urutan pengamatan. Masalahnya adalah algoritme standar menjalankan setiap sampel secara bergantian, di seluruh jaringan.

Sebaliknya, kita cukup menjalankan semua sampel N secara bersamaan melalui DBN, satu irisan pada satu waktu. Algoritme yang dimodifikasi sesuai dengan pola umum algoritme penyaringan, dengan kumpulan sampel N sebagai pesan penerusan. Inovasi kunci pertama, kemudian, adalah menggunakan sampel itu sendiri sebagai representasi perkiraan dari distribusi status terkini. Ini memenuhi persyaratan waktu "konstan" per pembaruan, meskipun konstanta tersebut bergantung pada jumlah sampel yang diperlukan untuk mempertahankan perkiraan yang akurat. Tidak perlu juga mengurai DBN, karena kita hanya perlu memiliki irisan terkini dan irisan berikutnya dalam memori.

Dalam pembahasan kita tentang pembobotan, kami menunjukkan bahwa akurasi algoritme menurun jika variabel bukti berada "di hilir" dari variabel yang sedang diambil sampelnya, karena dalam kasus tersebut sampel dihasilkan tanpa pengaruh apa pun dari bukti. Melihat struktur khas DBN—misalnya, DBN payung pada Gambar 6.16—kita melihat bahwa memang variabel status awal akan diambil sampelnya tanpa manfaat dari bukti selanjutnya. Faktanya, jika kita melihat lebih cermat, kita melihat bahwa tidak ada variabel status yang memiliki variabel bukti di antara leluhurnya! Oleh karena itu, meskipun bobot setiap sampel akan bergantung pada bukti, kumpulan sampel aktual yang dihasilkan akan sepenuhnya independen dari bukti. Misalnya, meskipun bos membawa payung setiap hari, proses pengambilan sampel masih dapat berhalusinasi tentang hari-hari cerah yang tak berujung.

Artinya dalam praktiknya adalah bahwa fraksi sampel yang tetap cukup dekat dengan rangkaian kejadian aktual (dan karenanya memiliki bobot yang tidak dapat diabaikan) turun secara eksponensial dengan t , panjang urutan pengamatan. Dengan kata lain, untuk mempertahankan tingkat akurasi tertentu, kita perlu meningkatkan jumlah sampel secara eksponensial dengan t .

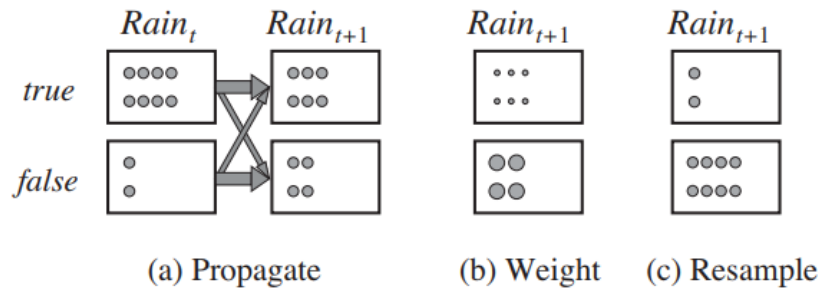
Mengingat bahwa algoritme penyaringan yang bekerja secara real time hanya dapat menggunakan sejumlah sampel yang tetap, yang terjadi dalam praktiknya adalah bahwa kesalahan tersebut meledak setelah sejumlah langkah pembaruan yang sangat kecil. Jelas, kita membutuhkan solusi yang lebih baik. Inovasi kunci kedua adalah memfokuskan kumpulan sampel pada wilayah probabilitas tinggi dari ruang status.

Hal ini dapat dilakukan dengan membuang sampel yang memiliki bobot sangat rendah, menurut pengamatan, sambil mereplikasi sampel yang memiliki bobot tinggi. Dengan cara itu, populasi sampel akan tetap cukup dekat dengan kenyataan. Jika kita menganggap sampel sebagai sumber daya untuk memodelkan distribusi posterior, maka masuk akal untuk menggunakan lebih banyak sampel di wilayah ruang keadaan tempat posterior lebih tinggi.

Sekelompok algoritme yang disebut penyaringan partikel dirancang untuk melakukan hal itu. Penyaringan partikel bekerja sebagai berikut: Pertama, populasi sampel keadaan awal N dibuat dengan mengambil sampel dari distribusi sebelumnya $P(X_0)$. Kemudian siklus pembaruan diulang untuk setiap langkah waktu:

1. Setiap sampel disebarkan ke depan dengan mengambil sampel nilai keadaan berikutnya x_{t+1} yang diberikan nilai saat ini x_t untuk sampel, berdasarkan model transisi $P(X_{t+1} | x_t)$.
2. Setiap sampel dibobot dengan kemungkinan yang diberikannya pada bukti baru, $P(e_{t+1} | x_{t+1})$.
3. Populasi diambil sampelnya kembali untuk menghasilkan populasi baru dari sampel N . Setiap sampel baru dipilih dari populasi saat ini; probabilitas bahwa sampel tertentu dipilih sebanding dengan bobotnya. Sampel baru tidak diberi bobot.

Algoritme ditunjukkan secara rinci pada Gambar 6.17, dan operasinya untuk DBN payung diilustrasikan pada Gambar 6.18.



Gambar 6.18 Siklus pembaruan penyingkapan partikel untuk DBN payung dengan $N = 10$, yang menunjukkan populasi sampel setiap keadaan. (a) Pada waktu t , 8 sampel mengindikasikan hujan dan 2 mengindikasikan hujan. Masing-masing disebarkan ke depan dengan mengambil sampel keadaan berikutnya melalui model transisi. Pada waktu $t + 1$, 6 sampel mengindikasikan hujan dan 4 mengindikasikan hujan. (b) Payung diamati pada $t + 1$. Setiap sampel dibobot berdasarkan kemungkinannya untuk pengamatan, seperti yang ditunjukkan oleh ukuran lingkaran. (c) Satu set baru yang terdiri dari 10 sampel dihasilkan oleh pemilihan acak terbobot dari set saat ini, menghasilkan 2 sampel yang mengindikasikan hujan dan 8 yang mengindikasikan hujan.

Kita dapat menunjukkan bahwa algoritma ini konsisten—memberikan probabilitas yang benar saat N cenderung tak terhingga—dengan mempertimbangkan apa yang terjadi selama satu siklus pembaruan. Kita berasumsi bahwa populasi sampel dimulai dengan representasi yang benar dari pesan penerusan $f_{1:t} = P(X_t | e_{1:t})$ pada waktu t . Dengan menuliskan $N(x_t | e_{1:t})$ untuk jumlah sampel yang menempati status x_t setelah pengamatan $e_{1:t}$ diproses, maka kita memiliki:

$$N(x_t | e_{1:t})/N = P(x_t | e_{1:t}) \quad (6.23)$$

untuk N yang besar. Sekarang kita menyebarkan setiap sampel ke depan dengan mengambil sampel variabel keadaan pada $t + 1$, dengan nilai sampel pada t yang diberikan. Jumlah sampel yang mencapai keadaan x_{t+1} dari setiap x_t adalah probabilitas transisi dikalikan populasi x_t ; oleh karena itu, jumlah total sampel yang mencapai x_{t+1} adalah:

$$N(x_{t+1} | e_{1:t}) = \sum_{x_t} P(x_{t+1} | x_t) N(x_t | e_{1:t}) .$$

Sekarang kita beri bobot pada setiap sampel berdasarkan kemungkinan bukti pada $t + 1$. Sampel dalam keadaan x_{t+1} menerima bobot $P(e_{t+1} | x_{t+1})$. Oleh karena itu, bobot total sampel dalam x_{t+1} setelah melihat e_{t+1} adalah:

$$W(x_{t+1} | e_{1:t+1}) = P(e_{t+1} | x_{t+1}) N(x_{t+1} | e_{1:t}) .$$

Sekarang untuk langkah resampling. Karena setiap sampel direplikasi dengan probabilitas yang proporsional terhadap bobotnya, jumlah sampel dalam keadaan x_{t+1} setelah resampling proporsional terhadap bobot total dalam x_{t+1} sebelum resampling:

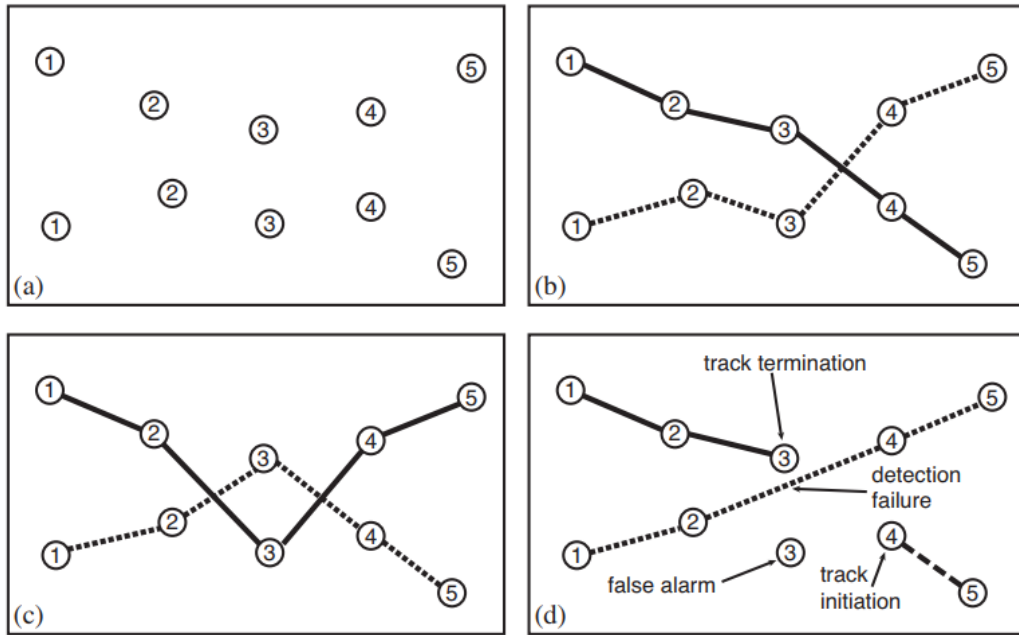
$$\begin{aligned}
 N(x_{t+1} | e_{1:t+1})/N &= \alpha W(x_{t+1} | e_{1:t+1}) \\
 &= \alpha P(e_{t+1} | x_{t+1})N(x_{t+1} | e_{1:t}) \\
 &= \alpha P(e_{t+1} | x_{t+1}) \sum_{x_t} P(x_{t+1} | x_t)N(x_t | e_{1:t}) \\
 &= \alpha NP(e_{t+1} | x_{t+1}) \sum_{x_t} P(x_{t+1} | x_t)P(x_t | e_{1:t}) \quad (by\ 6.23) \\
 &= \alpha' P(e_{t+1} | x_{t+1}) \sum_{x_t} P(x_{t+1} | x_t)P(x_t | e_{1:t}) \\
 &= P(x_{t+1} | e_{1:t+1}) \quad (by\ 15.5).
 \end{aligned}$$

Oleh karena itu, populasi sampel setelah satu siklus pembaruan dengan tepat merepresentasikan pesan maju pada waktu $t + 1$.

Oleh karena itu, penyaringan partikel konsisten, tetapi apakah efisien? Dalam praktiknya, tampaknya jawabannya adalah ya: penyaringan partikel tampaknya mempertahankan perkiraan yang baik terhadap posterior sebenarnya menggunakan jumlah sampel yang konstan. Di bawah asumsi tertentu—khususnya, bahwa probabilitas dalam model transisi dan sensor secara ketat lebih besar dari 0 dan kurang dari 1—adalah mungkin untuk membuktikan bahwa perkiraan tersebut mempertahankan kesalahan terbatas dengan probabilitas tinggi. Di sisi praktis, rentang aplikasi telah berkembang hingga mencakup banyak bidang sains dan teknik; beberapa referensi diberikan di akhir bab.

6.6 MELACAK BANYAK OBJEK

Bagian sebelumnya telah mempertimbangkan—tanpa menyebutkannya—masalah estimasi keadaan yang melibatkan satu objek. Di bagian ini, kita melihat apa yang terjadi ketika dua objek atau lebih menghasilkan pengamatan. Yang membuat kasus ini berbeda dari estimasi keadaan lama yang biasa adalah bahwa sekarang ada kemungkinan ketidakpastian tentang objek mana yang menghasilkan observasi mana. Ini adalah masalah ketidakpastian identitas, yang sekarang dilihat dalam konteks temporal. Dalam literatur teori kontrol, ini adalah masalah asosiasi data—yaitu, masalah mengaitkan data observasi dengan objek yang menghasilkannya.



Gambar 6.19 (a) Pengamatan yang dilakukan terhadap lokasi objek dalam ruang 2D selama lima langkah waktu. Setiap pengamatan diberi label dengan langkah waktu tetapi tidak mengidentifikasi objek yang menghasilkannya. (b–c) Hipotesis yang mungkin tentang jejak objek yang mendasarinya. (d) Hipotesis untuk kasus di mana alarm palsu, kegagalan deteksi, dan inisiasi/penghentian jejak mungkin terjadi.

Masalah asosiasi data awalnya dipelajari dalam konteks pelacakan radar, di mana pulsa pantulan dideteksi pada interval waktu tetap oleh antena radar yang berputar. Pada setiap langkah waktu, beberapa titik mungkin muncul di layar, tetapi tidak ada pengamatan langsung tentang titik mana pada waktu t yang termasuk dalam titik mana pada waktu $t - 1$. Gambar 6.19(a) menunjukkan contoh sederhana dengan dua titik per langkah waktu selama lima langkah. Misalkan dua lokasi titik pada waktu t adalah e_t^1 dan e_t^2 . (Pelabelan titik dalam langkah waktu sebagai "1" dan "2" sepenuhnya sembarangan dan tidak mengandung informasi apa pun.) Mari kita asumsikan, untuk saat ini, bahwa tepat dua pesawat, A dan B, menghasilkan titik-titik tersebut; posisi sebenarnya adalah X_t^A dan X_t^B . Agar semuanya tetap sederhana, kita juga akan mengasumsikan bahwa setiap pesawat bergerak secara independen menurut model transisi yang diketahui—misalnya, model Gaussian linier seperti yang digunakan dalam filter Kalman. Misalkan kita mencoba menuliskan model probabilitas keseluruhan untuk skenario ini, seperti yang kita lakukan untuk proses temporal umum dalam Persamaan (6.3). Seperti biasa, faktor distribusi gabungan menjadi kontribusi untuk setiap langkah waktu sebagai berikut:

$$P(x_{0:t}^A, x_{0:t}^B, e_{1:t}^1, e_{1:t}^2) = P(x_0^A)P(x_0^B) \prod_{i=1}^t P(x_i^A | x_{i-1}^A)P(x_i^B | x_{i-1}^B)P(e_i^1, e_i^2 | x_i^A, x_i^B). \quad (6.24)$$

Kita ingin memfaktorkan suku pengamatan $P(e_i^1, e_i^2 | x_i^A, x_i^B)$ menjadi hasil perkalian dua suku, satu untuk setiap objek, tetapi ini memerlukan pengetahuan tentang pengamatan mana yang dihasilkan oleh objek mana. Sebaliknya, kita harus menjumlahkan semua kemungkinan cara untuk mengaitkan pengamatan dengan objek. Beberapa cara tersebut ditunjukkan pada Gambar 6.19(b–c); secara umum, untuk n objek dan T langkah waktu, ada $(n!)^T$ cara untuk melakukannya—jumlah yang sangat besar.

Secara matematis, "cara mengaitkan pengamatan dengan objek" adalah kumpulan variabel acak yang tidak teramati yang mengidentifikasi sumber setiap pengamatan. Kita akan menulis ω_t untuk menunjukkan pemetaan satu-satu dari objek ke pengamatan pada waktu t , dengan $\omega_t(A)$ dan $\omega_t(B)$ menunjukkan pengamatan spesifik (1 atau 2) yang ditugaskan ω_t ke A dan B. (Untuk n objek, ω_t akan memiliki $n!$ nilai yang mungkin; di sini, $n! = 2$.) Karena label "1" dan "2" pada pengamatan ditugaskan secara sewenang-wenang, prior pada ω_t seragam dan ω_t independen dari status objek, x_t^A dan x_t^B). Jadi kita dapat mengkondisikan istilah pengamatan $P(e_i^1, e_i^2 | x_i^A, x_i^B)$ dan kemudian sederhanakan:

$$\begin{aligned} P(e_i^1, e_i^2 | x_i^A, x_i^B) &= \sum_{\omega_i} P(e_i^1, e_i^2 | x_i^A, x_i^B, \omega_i) P(\omega_i | x_i^A, x_i^B) \\ &= \sum_{\omega_i} P(e_i^{\omega_i(A)} | x_i^A) P(e_i^{\omega_i(B)} | x_i^B) P(\omega_i | x_i^A, x_i^B) \\ &= \frac{1}{2} \sum_{\omega_i} P(e_i^{\omega_i(A)} | x_i^A) P(e_i^{\omega_i(B)} | x_i^B). \end{aligned}$$

Dengan memasukkan ini ke Persamaan (6.24), kita memperoleh ekspresi yang hanya dalam bentuk model transisi dan sensor untuk objek dan pengamatan individual.

Seperti halnya semua model probabilitas, inferensi berarti menjumlahkan variabel selain kueri dan bukti. Untuk pemfilteran dalam HMM dan DBN, kita dapat menjumlahkan variabel status dari 1 hingga $t - 1$ dengan trik pemrograman dinamis sederhana; untuk filter Kalman, kita memanfaatkan properti khusus Gaussian. Untuk asosiasi data, kita kurang beruntung. Tidak ada algoritme eksak yang efisien (yang diketahui), karena alasan yang sama dengan tidak ada algoritme untuk filter Kalman switching: distribusi pemfilteran $P(x_t^A | e_{1:t}^1, e_{1:t}^2)$ untuk objek A berakhir sebagai campuran dari distribusi yang banyak secara eksponensial, satu untuk setiap cara memilih urutan pengamatan untuk ditetapkan ke A.

Sebagai hasil dari kompleksitas inferensi eksak, banyak metode perkiraan yang berbeda telah digunakan. Pendekatan yang paling sederhana adalah memilih satu penugasan "terbaik" pada setiap langkah waktu, dengan mempertimbangkan posisi objek yang diprediksi pada langkah waktu saat ini. Penugasan ini mengaitkan pengamatan dengan objek dan memungkinkan jalur setiap objek diperbarui dan prediksi dibuat untuk langkah waktu berikutnya. Untuk memilih penugasan "terbaik", biasanya digunakan apa yang disebut filter tetangga terdekat, yang berulang kali memilih pasangan terdekat dari posisi yang diprediksi dan pengamatan dan menambahkan pasangan itu ke penugasan.

Filter tetangga terdekat berfungsi dengan baik ketika objek terpisah dengan baik dalam ruang keadaan dan ketidakpastian prediksi dan kesalahan pengamatan kecil—dengan kata lain, ketika tidak ada kemungkinan kebingungan. Ketika ada lebih banyak ketidakpastian mengenai penugasan yang benar, pendekatan yang lebih baik adalah memilih penugasan yang memaksimalkan probabilitas gabungan dari pengamatan saat ini dengan mempertimbangkan posisi yang diprediksi. Ini dapat dilakukan dengan sangat efisien menggunakan algoritma Hungaria, meskipun ada $n!$ penugasan untuk dipilih.

Metode apa pun yang berkomitmen pada satu penugasan terbaik pada setiap langkah waktu gagal total dalam kondisi yang lebih sulit. Secara khusus, jika algoritme berkomitmen pada penugasan yang salah, prediksi pada langkah waktu berikutnya mungkin salah secara signifikan, yang mengarah ke lebih banyak penugasan yang salah, dan seterusnya. Dua pendekatan modern ternyata jauh lebih efektif.

Algoritma penyaringan partikel untuk asosiasi data bekerja dengan mempertahankan koleksi besar kemungkinan penugasan saat ini. Algoritma MCMC mengeksplorasi ruang riwayat penugasan—misalnya, Gambar 6.19(b–c) mungkin merupakan status dalam ruang status MCMC—dan dapat mengubah pikirannya tentang keputusan penugasan sebelumnya. Metode asosiasi data MCMC saat ini dapat menangani ratusan objek secara real time sambil memberikan perkiraan yang baik untuk distribusi posterior yang sebenarnya.



Gambar 6.20 Gambar dari kamera pengintai (a) di hulu dan (b) di hilir yang berjarak sekitar dua mil di Highway 99 di Sacramento, California. Kendaraan yang terbungkus kotak telah teridentifikasi di kedua kamera.

Skenario yang dijelaskan sejauh ini melibatkan n objek yang diketahui yang menghasilkan n pengamatan pada setiap langkah waktu. Aplikasi asosiasi data yang sebenarnya biasanya jauh lebih rumit. Sering kali, pengamatan yang dilaporkan mencakup alarm palsu (juga dikenal sebagai kekacauan), yang tidak disebabkan oleh objek nyata. Kegagalan deteksi dapat terjadi, yang berarti tidak ada pengamatan yang dilaporkan untuk objek nyata. Akhirnya, objek baru muncul dan objek lama menghilang. Fenomena ini, yang menciptakan lebih banyak kemungkinan dunia yang perlu dikhawatirkan, diilustrasikan dalam Gambar 6.19(d).

Gambar 6.20 menunjukkan dua gambar dari kamera yang terpisah jauh di jalan bebas hambatan California. Dalam aplikasi ini, kami tertarik pada dua tujuan: memperkirakan waktu yang dibutuhkan, dalam kondisi lalu lintas saat ini, untuk pergi dari satu tempat ke tempat lain di sistem jalan bebas hambatan; dan mengukur permintaan, yaitu, berapa banyak kendaraan

yang melakukan perjalanan antara dua titik mana pun dalam sistem pada waktu tertentu dalam sehari dan pada hari tertentu dalam seminggu. Kedua tujuan tersebut memerlukan penyelesaian masalah asosiasi data di area yang luas dengan banyak kamera dan puluhan ribu kendaraan per jam.

Dengan pengawasan visual, alarm palsu disebabkan oleh bayangan yang bergerak, kendaraan yang bergerak, pantulan di genangan air, dll.; kegagalan deteksi disebabkan oleh oklusi, kabut, kegelapan, dan kurangnya kontras visual; dan kendaraan terus-menerus masuk dan keluar dari sistem jalan bebas hambatan. Lebih jauh lagi, tampilan kendaraan apa pun dapat berubah secara dramatis di antara kamera tergantung pada kondisi pencahayaan dan pose kendaraan dalam gambar, dan model transisi berubah saat kemacetan lalu lintas datang dan pergi. Meskipun ada masalah ini, algoritma asosiasi data modern telah berhasil memperkirakan parameter lalu lintas dalam pengaturan dunia nyata. Secara tegas, distribusi Gaussian bermasalah karena menetapkan probabilitas bukan nol pada tingkat muatan negatif yang besar. Distribusi beta terkadang merupakan pilihan yang lebih baik untuk variabel yang jangkauannya terbatas.

Asosiasi data merupakan fondasi penting untuk melacak dunia yang kompleks, karena tanpanya tidak ada cara untuk menggabungkan beberapa pengamatan dari objek tertentu. Ketika objek di dunia berinteraksi satu sama lain dalam aktivitas yang kompleks, memahami dunia memerlukan penggabungan asosiasi data dengan model probabilitas relasional dan alam semesta terbuka. Ini saat ini merupakan area penelitian yang aktif.

6.7 RINGKASAN

Bab ini telah membahas masalah umum dalam merepresentasikan dan menalar proses temporal probabilistik. Poin-poin utamanya adalah sebagai berikut:

- ❖ Keadaan dunia yang berubah ditangani dengan menggunakan serangkaian variabel acak untuk merepresentasikan keadaan pada setiap titik waktu.
- ❖ Representasi dapat dirancang untuk memenuhi properti Markov, sehingga masa depan tidak bergantung pada masa lalu mengingat masa kini. Dikombinasikan dengan asumsi bahwa proses tersebut stasioner—yaitu, dinamika tidak berubah seiring waktu—ini sangat menyederhanakan representasi.
- ❖ Model probabilitas temporal dapat dianggap berisi model transisi yang menggambarkan evolusi status dan model sensor yang menggambarkan proses observasi.
- ❖ Tugas inferensi utama dalam model temporal adalah penyaringan, prediksi, penghalusan, dan penghitungan penjelasan yang paling mungkin. Masing-masing dapat dicapai dengan menggunakan algoritma rekursif sederhana yang waktu prosesnya linier dalam panjang urutan.
- ❖ Tiga keluarga model temporal dipelajari secara lebih mendalam: model Markov tersembunyi, filter Kalman, dan jaringan Bayesian dinamis (yang menyertakan dua lainnya sebagai kasus khusus).

- ❖ Kecuali jika asumsi khusus dibuat, seperti dalam filter Kalman, inferensi eksak dengan banyak variabel status tidak dapat dilakukan. Dalam praktiknya, algoritma penyaringan partikel tampaknya merupakan algoritma perkiraan yang efektif.
- ❖ Ketika mencoba melacak banyak objek, ketidakpastian muncul mengenai observasi mana yang termasuk dalam objek mana—masalah asosiasi data. Jumlah hipotesis asosiasi biasanya sangat besar, tetapi algoritma MCMC dan penyaringan partikel untuk asosiasi data berfungsi dengan baik dalam praktik.

BAB 7

MEMBUAT KEPUTUSAN SEDERHANA

7.1 MENGGABUNGKAN KEYAKINAN DAN KEINGINAN DALAM KETIDAKPASTIAN

Teori keputusan, dalam bentuknya yang paling sederhana, membahas pemilihan di antara tindakan berdasarkan keinginan akan hasil langsungnya; yaitu, lingkungan diasumsikan bersifat episodik.

Kami menggunakan notasi $RESULT(s_0, a)$ untuk keadaan yang merupakan hasil deterministik dari mengambil tindakan a dalam keadaan s_0 . Dalam bab ini, kami membahas lingkungan yang dapat diamati sebagian dan tidak deterministik. Karena agen mungkin tidak mengetahui keadaan saat ini, kita menghilangkannya dan mendefinisikan $RESULT(a)$ sebagai variabel acak yang nilainya merupakan kemungkinan keadaan hasil. Probabilitas hasil s' , dengan pengamatan bukti e , ditulis:

$$P(RESULT(a) = s' \mid a, e),$$

di mana a pada sisi kanan bilah pengkondisian menunjukkan peristiwa bahwa tindakan a dieksekusi.

Preferensi agen ditangkap oleh fungsi utilitas, $U(s)$, yang menetapkan satu angka untuk mengekspresikan keinginan akan suatu keadaan. Utilitas yang diharapkan dari suatu tindakan berdasarkan bukti, $EU(a \mid e)$, hanyalah nilai utilitas rata-rata dari hasil, yang dibobot dengan probabilitas bahwa hasil tersebut terjadi:

$$EU(a \mid e) = \sum_{s'} P(RESULT(a) = s' \mid a, e) U(s') \quad (7.1)$$

Prinsip utilitas harapan maksimum (MEU) menyatakan bahwa agen rasional harus memilih tindakan yang memaksimalkan utilitas harapan agen:

$$action = \underset{a}{argmax} EU(a \mid e)$$

Dalam arti tertentu, prinsip MEU dapat dilihat sebagai pendefinisian seluruh AI. Yang harus dilakukan agen cerdas adalah menghitung berbagai kuantitas, memaksimalkan utilitas atas tindakannya, dan selesai. Namun, ini tidak berarti bahwa masalah AI diselesaikan dengan definisi!

Prinsip MEU memformalkan gagasan umum bahwa agen harus "melakukan hal yang benar," tetapi hanya sedikit melangkah menuju operasionalisasi penuh dari saran tersebut. Memperkirakan keadaan dunia memerlukan persepsi, pembelajaran, representasi

pengetahuan, dan inferensi. Menghitung $P(HASIL(a) \mid a, e)$ memerlukan model kausal dunia yang lengkap, inferensi NP-hard dalam jaringan Bayesian (yang sangat besar). Menghitung utilitas hasil $U(s')$ sering kali memerlukan pencarian atau perencanaan, karena agen mungkin tidak mengetahui seberapa baik suatu keadaan hingga ia mengetahui ke mana ia dapat mencapai keadaan tersebut. Jadi, teori keputusan bukanlah obat mujarab yang memecahkan masalah AI—tetapi teori ini menyediakan kerangka kerja yang berguna. Prinsip MEU memiliki hubungan yang jelas dengan gagasan tentang ukuran kinerja.

Gagasan dasarnya sederhana. Pertimbangkan lingkungan yang dapat menyebabkan agen memiliki riwayat persepsi tertentu, dan pertimbangkan berbagai agen yang dapat kita rancang. Jika agen bertindak untuk memaksimalkan fungsi utilitas yang secara tepat mencerminkan ukuran kinerja, maka agen akan mencapai skor kinerja setinggi mungkin (diratakan dari semua lingkungan yang memungkinkan). Ini adalah pembenaran utama untuk prinsip MEU itu sendiri. Meskipun klaim tersebut mungkin tampak tautologis, klaim tersebut sebenarnya mewujudkan transisi yang sangat penting dari kriteria rasionalitas global dan eksternal—ukuran kinerja atas riwayat lingkungan—ke kriteria lokal dan internal yang melibatkan pemaksimalan fungsi utilitas yang diterapkan pada keadaan berikutnya.

7.2 DASAR TEORI UTILITAS

Secara intuitif, prinsip *Maximum Expected Utility* (MEU) tampak seperti cara yang masuk akal untuk membuat keputusan, tetapi sama sekali tidak jelas bahwa itu adalah satu-satunya cara yang rasional. Lagi pula, mengapa memaksimalkan utilitas rata-rata menjadi begitu istimewa? Apa yang salah dengan agen yang memaksimalkan jumlah tertimbang dari pangkat tiga utilitas yang mungkin, atau mencoba meminimalkan kerugian terburuk yang mungkin terjadi? Bisakah agen bertindak secara rasional hanya dengan mengekspresikan preferensi antarnegara, tanpa memberi mereka nilai numerik? Terakhir, mengapa fungsi utilitas dengan properti yang diperlukan harus ada?

Kendala pada Preferensi Rasional

Pertanyaan-pertanyaan ini dapat dijawab dengan menuliskan beberapa kendala pada preferensi yang seharusnya dimiliki oleh agen rasional dan kemudian menunjukkan bahwa prinsip MEU dapat diturunkan dari kendala tersebut. Kami menggunakan notasi berikut untuk menggambarkan preferensi agen:

$A \succ B$, agen lebih menyukai A daripada B .

$A \sim B$, agen acuh tak acuh antara A dan B .

$A \succeq B$, agen lebih menyukai A daripada B atau acuh tak acuh di antara keduanya.

Sekarang pertanyaan yang jelas adalah, apa saja hal-hal yang merupakan A dan B ? Mereka bisa saja merupakan keadaan dunia, tetapi seringkali terdapat ketidakpastian tentang apa yang sebenarnya ditawarkan. Misalnya, seorang penumpang pesawat yang ditawari "hidangan pasta atau ayam" tidak tahu apa yang tersembunyi di balik lapisan kertas timah. Pastanya bisa saja lezat atau beku, ayamnya berair atau terlalu matang hingga tak dikenali. Kita dapat menganggap serangkaian hasil untuk setiap tindakan sebagai lotere—anggaplah setiap

tindakan sebagai sebuah tiket. Lotere L dengan kemungkinan hasil $S_1, \dots, S_n$ yang terjadi dengan probabilitas $p_1, \dots, p_n$ ditulis

$$L = [p_1, S_1; p_2, S_2; \dots p_n, S_n] .$$

Secara umum, setiap hasil S_i dari lotere dapat berupa keadaan atom atau lotere lainnya. Masalah utama bagi teori utilitas adalah memahami bagaimana preferensi antara lotere kompleks terkait dengan preferensi antara keadaan dasar dalam lotere tersebut. Untuk mengatasi masalah ini, kami mencantumkan enam kendala yang harus dipatuhi oleh setiap hubungan preferensi yang wajar:

- Keteraturan: Diberikan dua lotere, agen rasional harus lebih memilih satu daripada yang lain atau menilai keduanya sama-sama lebih disukai. Artinya, agen tidak dapat menghindari keputusan. Menolak bertaruh sama seperti menolak membiarkan waktu berlalu.

Tepat satu dari $(A \succ B)$, $(B \succ A)$ atau $(A \sim B)$ yang berlaku.

- Transitivitas: Diberikan tiga lotere, jika seorang agen lebih menyukai A daripada B dan lebih menyukai B daripada C , maka agen tersebut harus lebih menyukai A daripada C .

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C) .$$

- Kontinuitas: Jika suatu lotere B lebih disukai antara A dan C , maka ada probabilitas p di mana agen rasional akan bersikap acuh tak acuh antara mendapatkan B secara pasti dan lotere yang menghasilkan A dengan probabilitas p dan C dengan probabilitas $1 - p$.

$$A \succ B \succ C \Rightarrow \exists_p [p, A; 1 - p, C] \sim B .$$

- Substitusi: Jika seorang agen bersikap acuh tak acuh antara dua lotere A dan B , maka agen tersebut bersikap acuh tak acuh antara dua lotere yang lebih kompleks yang sama, kecuali bahwa B disubstitusikan untuk A di salah satunya. Hal ini berlaku terlepas dari probabilitas dan hasil lainnya dalam lotere tersebut.

$$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]$$

Hal ini juga berlaku jika kita mengganti $\succ$ dengan $\sim$ dalam aksioma ini.

- Monotonisitas: Misalkan dua lotere memiliki dua kemungkinan hasil yang sama, A dan B . Jika seorang agen lebih menyukai A daripada B , maka agen tersebut harus lebih menyukai lotere yang memiliki probabilitas lebih tinggi untuk A (dan sebaliknya).

$$A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1 - p, B] \succ [q, A; 1 - q, B]) .$$

- Kemampuan terurai: Lotere majemuk dapat disederhanakan menjadi lotere yang lebih sederhana menggunakan hukum probabilitas. Ini disebut aturan “tidak menyenangkan dalam perjudian” karena aturan ini menyatakan bahwa dua lotere yang berurutan dapat dipadatkan menjadi satu lotere yang setara, seperti yang ditunjukkan pada Gambar 7.1(b).

$$[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q), C] .$$

Kendala-kendala ini dikenal sebagai aksioma teori utilitas. Setiap aksioma dapat dimotivasi dengan menunjukkan bahwa agen yang melanggarnya akan menunjukkan perilaku yang jelas-jelas irasional dalam beberapa situasi. Misalnya, kita dapat memotivasi transitivitas dengan membuat agen dengan preferensi nontransitif memberi kita semua uangnya.

Misalkan agen tersebut memiliki preferensi nontransitif $A \succ B \succ C \succ A$, di mana A , B , dan C adalah barang yang dapat dipertukarkan secara bebas. Jika agen saat ini memiliki A , maka kita dapat menawarkan untuk menukar C dengan A ditambah satu sen. Agen tersebut lebih menyukai C , sehingga bersedia melakukan pertukaran ini. Kita kemudian dapat menawarkan untuk menukar B dengan C , mengambil satu sen lagi, dan akhirnya menukar A dengan B . Ini membawa kita kembali ke titik awal, kecuali bahwa agen tersebut telah memberi kita tiga sen (Gambar 7.1(a)). Kita dapat terus berputar dalam siklus tersebut hingga agen tersebut tidak memiliki uang sama sekali. Jelas, agen tersebut telah bertindak irasional dalam kasus ini.

Preferensi Mengarah pada Utilitas

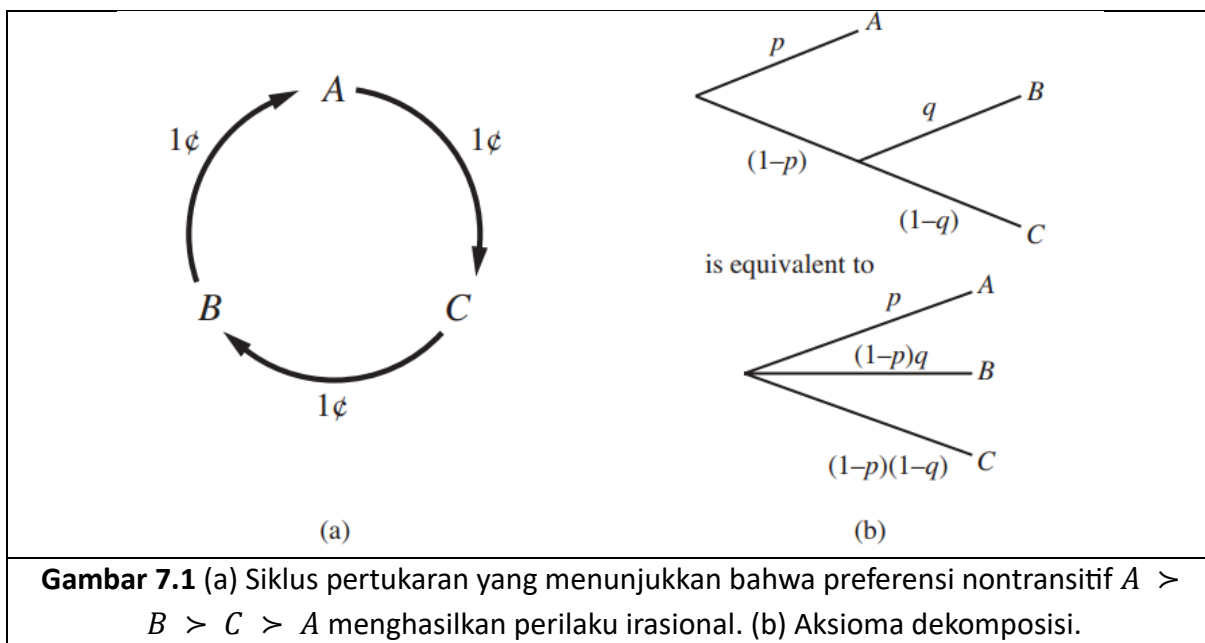
Perhatikan bahwa aksioma-aksioma teori utilitas sebenarnya adalah aksioma tentang preferensi—aksioma-aksioma tersebut tidak menjelaskan apa pun tentang fungsi utilitas. Namun, dari aksioma-aksioma utilitas tersebut, kita dapat menyimpulkan konsekuensi-konsekuensi berikut:

- Keberadaan Fungsi Utilitas: Jika preferensi suatu agen mematuhi aksioma utilitas, maka terdapat suatu fungsi U sehingga $U(A) > U(B)$ jika dan hanya jika A lebih disukai daripada B , dan $U(A) = U(B)$ jika dan hanya jika agen tersebut indifferent antara A dan B .

$$\begin{aligned} U(A) > U(B) &\Leftrightarrow A \succ B \\ U(A) = U(B) &\Leftrightarrow A \sim B \end{aligned}$$

- Utilitas yang Diharapkan dari Lotere: Utilitas lotere adalah jumlah probabilitas setiap hasil dikalikan utilitas hasil tersebut.

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$



Dengan kata lain, setelah probabilitas dan utilitas dari kemungkinan hasil ditentukan, utilitas lotere gabungan yang melibatkan keadaan tersebut ditentukan sepenuhnya. Karena hasil dari tindakan nondeterministik adalah lotere, maka agen dapat bertindak rasional—yaitu, konsisten dengan preferensinya—hanya dengan memilih tindakan yang memaksimalkan utilitas yang diharapkan menurut Persamaan (7.1).

Teorema sebelumnya menetapkan bahwa fungsi utilitas ada untuk agen rasional apa pun, tetapi tidak menetapkan bahwa fungsi tersebut unik. Faktanya, mudah untuk melihat bahwa perilaku agen tidak akan berubah jika fungsi utilitasnya $U(S)$ diubah menurut

$$U'(S) = aU(S) + b, \quad (7.2)$$

di mana a dan b adalah konstanta dan $a > 0$; transformasi afin. Di sini, kita melihat bahwa hal ini sepenuhnya umum.

Seperti dalam permainan, dalam lingkungan deterministik, agen hanya perlu peringkat preferensi pada status—angka tidak penting. Ini disebut fungsi nilai atau fungsi utilitas ordinal. Penting untuk diingat bahwa keberadaan fungsi utilitas yang menggambarkan perilaku preferensi agen tidak selalu berarti bahwa agen secara eksplisit memaksimalkan fungsi utilitas tersebut dalam pertimbangannya sendiri. Perilaku rasional dapat dihasilkan dalam sejumlah cara. Namun, dengan mengamati preferensi agen rasional, seorang pengamat dapat membangun fungsi utilitas yang mewakili apa yang sebenarnya ingin dicapai oleh agen.

7.3 FUNGSI UTILITAS

Utilitas adalah fungsi yang memetakan dari lotere ke bilangan riil. Kita tahu ada beberapa aksioma tentang utilitas yang harus dipatuhi oleh semua agen rasional. Apakah hanya itu yang bisa kita katakan tentang fungsi utilitas? Singkatnya, hanya itu: seorang agen dapat memiliki preferensi apa pun yang disukainya. Misalnya, seorang agen mungkin lebih

suka memiliki sejumlah dolar di rekening banknya; dalam hal ini, jika ia memiliki Rp 160.000, ia akan memberikan Rp 30.000.

Ini mungkin tidak biasa, tetapi kita tidak bisa menyebutnya irasional. Seorang agen mungkin lebih suka Ford Pinto tahun 1973 yang penyok daripada Mercedes baru yang mengkilap. Preferensi juga dapat berinteraksi: misalnya, agen mungkin lebih suka sejumlah dolar hanya ketika ia memiliki Pinto, tetapi ketika ia memiliki Mercedes, ia mungkin lebih suka lebih banyak dolar daripada lebih sedikit. Untungnya, preferensi agen riil biasanya lebih sistematis, sehingga lebih mudah ditangani.

Penilaian Utilitas dan Skala Utilitas

Jika kita ingin membangun sistem teoretis-keputusan yang membantu agen membuat keputusan atau bertindak atas namanya sendiri, pertama-tama kita harus menentukan fungsi utilitas agen tersebut. Proses ini, yang sering disebut elisitasi preferensi, melibatkan penyajian pilihan kepada agen dan menggunakan preferensi yang teramati untuk menentukan fungsi utilitas yang mendasarinya.

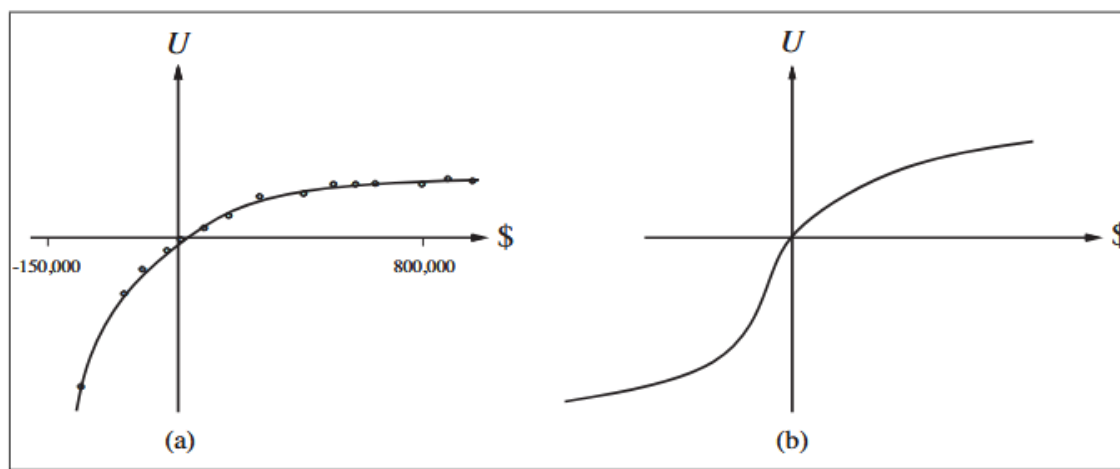
Persamaan (7.2) menyatakan bahwa tidak ada skala absolut untuk utilitas, tetapi tetap bermanfaat untuk menetapkan skala tertentu di mana utilitas dapat dicatat dan dibandingkan untuk setiap masalah tertentu. Skala dapat ditetapkan dengan menetapkan utilitas dari dua hasil tertentu, seperti halnya kita menetapkan skala suhu dengan menetapkan titik beku dan titik didih air.

Biasanya, kita menetapkan utilitas "hadiah terbaik" pada $U(S) = u_{\top}$ dan "bencana terburuk" pada $U(S) = u_{\perp}$. Utilitas yang dinormalkan menggunakan skala dengan $u_{\perp} = 0$ dan $u_{\top} = 1$. Dengan skala utilitas antara $u_{\top}$ dan $u_{\perp}$, kita dapat menilai utilitas hadiah tertentu S dengan meminta agen untuk memilih antara S dan lotere standar $[p, u_{\top}; (1 - p), u_{\perp}]$. Probabilitas p disesuaikan hingga agen bersikap acuh tak acuh antara S dan lotere standar. Dengan asumsi utilitas yang dinormalkan, utilitas S diberikan oleh p . Setelah ini dilakukan untuk setiap hadiah, utilitas untuk semua lotere yang melibatkan hadiah tersebut ditentukan.

Dalam masalah pengambilan keputusan medis, transportasi, dan lingkungan, antara lain, nyawa manusia dipertaruhkan. Dalam kasus seperti itu, $u_{\perp}$ adalah nilai yang ditetapkan untuk kematian langsung (atau mungkin banyak kematian). Meskipun tidak ada yang merasa nyaman untuk memberi nilai pada kehidupan manusia, faktanya adalah bahwa pertukaran selalu dilakukan. Pesawat udara dirombak total pada interval yang ditentukan oleh perjalanan dan mil yang diterbangkan, alih-alih setelah setiap perjalanan.

Mobil diproduksi dengan cara yang mengorbankan biaya dengan tingkat keselamatan kecelakaan. Paradoksnya, penolakan untuk "memberikan nilai moneter pada kehidupan" berarti bahwa kehidupan sering kali diremehkan. Ross Shachter menceritakan pengalamannya dengan sebuah badan pemerintah yang menugaskan sebuah studi tentang menyingkirkan asbes dari sekolah. Para analis keputusan yang melakukan studi tersebut mengasumsikan nilai dolar tertentu untuk kehidupan seorang anak usia sekolah, dan berpendapat bahwa pilihan rasional berdasarkan asumsi itu adalah menyingkirkan asbes. Badan tersebut, yang secara moral marah pada gagasan untuk menetapkan nilai kehidupan, menolak laporan itu begitu saja. Kemudian memutuskan untuk tidak menyingkirkan asbes—secara implisit menegaskan

nilai yang lebih rendah untuk kehidupan seorang anak daripada yang ditetapkan oleh para analis.



Gambar 7.2 Kegunaan uang. (a) Data empiris untuk Tn. Beard pada rentang terbatas. (b) Kurva khas untuk rentang penuh.

Beberapa upaya telah dilakukan untuk mencari tahu nilai yang diberikan orang pada kehidupan mereka sendiri. Salah satu "mata uang" umum yang digunakan dalam analisis medis dan keselamatan adalah micromort, peluang kematian satu banding sejuta. Jika Anda bertanya kepada orang-orang berapa banyak yang akan mereka bayar untuk menghindari risiko—misalnya, untuk menghindari bermain rolet Rusia dengan revolver laras sejuta—mereka akan menjawab dengan angka yang sangat besar, mungkin puluhan ribu dolar, tetapi perilaku mereka yang sebenarnya mencerminkan nilai moneter yang jauh lebih rendah untuk micromort.

Misalnya, mengendarai mobil sejauh 230 mil menimbulkan risiko satu micromort; selama masa pakai mobil Anda—misalnya, 92.000 mil—itu berarti 400 micromort. Orang-orang tampaknya bersedia membayar sekitar Rp 100.000.000 (dengan harga tahun 2009) lebih banyak untuk mobil yang lebih aman yang mengurangi setengah risiko kematian, atau sekitar Rp 500.000 per micromort. Sejumlah penelitian telah mengonfirmasi angka dalam kisaran ini pada banyak individu dan jenis risiko. Tentu saja, argumen ini hanya berlaku untuk risiko kecil. Kebanyakan orang tidak akan setuju untuk bunuh diri demi Rp 500 miliar.

Ukuran lainnya adalah QALY, atau tahun kehidupan yang disesuaikan dengan kualitas. Pasien dengan disabilitas bersedia menerima harapan hidup yang lebih pendek untuk dipulihkan ke kesehatan penuh. Misalnya, pasien ginjal rata-rata tidak peduli antara hidup dua tahun dengan mesin dialisis dan satu tahun dengan kesehatan penuh.

Kegunaan Uang

Teori utilitas berakar pada ekonomi, dan ekonomi menyediakan satu kandidat yang jelas untuk ukuran utilitas: uang (atau lebih khusus lagi, total aset bersih agen). Pertukaran uang yang hampir universal untuk semua jenis barang dan jasa menunjukkan bahwa uang memainkan peran penting dalam fungsi utilitas manusia. Biasanya agen lebih suka lebih

banyak uang daripada lebih sedikit, jika semua hal lain sama. Kita katakan bahwa agen menunjukkan preferensi monoton untuk lebih banyak uang. Ini tidak berarti bahwa uang berperilaku sebagai fungsi utilitas, karena tidak mengatakan apa pun tentang preferensi antara lotere yang melibatkan uang.

Misalkan Anda telah menang atas pesaing lain dalam acara permainan televisi. Pembawa acara kini menawarkan pilihan kepada Anda: Anda dapat mengambil hadiah Rp 10 miliar atau mempertaruhkannya dengan melempar koin. Jika koin muncul sisi kepala, Anda tidak akan mendapatkan apa pun, tetapi jika muncul sisi ekor, Anda akan mendapatkan Rp 25 miliar. Jika Anda seperti kebanyakan orang, Anda akan menolak taruhan dan mengantongi satu juta. Apakah Anda bersikap tidak rasional?

Dengan asumsi koin tersebut adil, nilai moneter yang diharapkan (EMV) dari taruhan tersebut adalah $\frac{1}{2}(\text{Rp}0) + \frac{1}{2}(\text{Rp } 25 \text{ miliar}) = \text{Rp } 12,5 \text{ miliar}$, yang lebih besar dari Rp 10 miliar awal. Namun, hal itu tidak berarti bahwa menerima taruhan merupakan keputusan yang lebih baik. Misalkan kita menggunakan S_n untuk menunjukkan keadaan memiliki kekayaan total Rpn , dan kekayaan Anda saat ini adalah Rpk . Maka utilitas yang diharapkan dari dua tindakan menerima dan menolak taruhan adalah

$$\begin{aligned} EU(\text{Accept}) &= \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+25.000.000.000}), \\ EU(\text{Decline}) &= U(S_{k+10.000.000.000}). \end{aligned}$$

Untuk menentukan apa yang harus dilakukan, kita perlu menetapkan utilitas pada kondisi hasil. Utilitas tidak berbanding lurus dengan nilai moneter, karena utilitas untuk satu juta pertama Anda sangat tinggi (atau begitulah kata mereka), sedangkan utilitas untuk satu juta tambahan lebih kecil.

Misalkan Anda menetapkan utilitas 5 untuk status keuangan Anda saat ini (S_k), 9 untuk kondisi $S_{k+2.500.000}$, dan 8 untuk kondisi $S_{k+1.000.000}$. Maka tindakan rasionalnya adalah menolak, karena utilitas yang diharapkan dari penerimaan hanya 7 (lebih kecil dari 8 untuk penolakan). Di sisi lain, seorang miliarder kemungkinan besar akan memiliki fungsi utilitas yang linier secara lokal pada kisaran beberapa juta lebih, dan dengan demikian akan menerima pertaruhan tersebut.

Dalam sebuah studi perintis tentang fungsi utilitas aktual, Grayson menemukan bahwa utilitas uang hampir sama persis dengan logaritma jumlahnya. (Ide ini pertama kali diusulkan oleh Bernoulli. Satu kurva utilitas tertentu, untuk seorang Tn. Beard, ditunjukkan pada Gambar 7.2(a). Data yang diperoleh untuk preferensi Tn. Beard konsisten dengan fungsi utilitas

$$U(S_{k+n}) = -263.31 + 22.09 \log(n + 1.500.000.000)$$

untuk rentang antara $n = -\text{Rp } 1.500.000.000$ dan $n = \text{Rp } 8.000.000.000$.

Kita tidak boleh berasumsi bahwa ini adalah fungsi utilitas definitif untuk nilai moneter, tetapi kemungkinan besar sebagian besar orang memiliki fungsi utilitas yang cekung untuk

kekayaan positif. Berutang itu buruk, tetapi preferensi antara berbagai tingkat utang dapat menunjukkan pembalikan cekung yang terkait dengan kekayaan positif. Misalnya, seseorang yang sudah berutang Rp 100 miliar mungkin akan menerima pertaruhan pada koin yang adil dengan keuntungan Rp 100 miliar untuk sisi kepala dan kerugian Rp 200 miliar untuk sisi ekor. Ini menghasilkan kurva berbentuk S yang ditunjukkan pada Gambar 7.2(b). Jika kita batasi perhatian kita pada bagian positif dari kurva, di mana kemiringannya menurun, maka untuk setiap lotere L , manfaat menghadapi lotere tersebut lebih kecil daripada manfaat menerima nilai moneter lotere yang diharapkan sebagai sesuatu yang pasti:

$$U(L) < U(S_{EMV(L)}).$$

Artinya, agen dengan kurva bentuk ini cenderung menghindari risiko: mereka lebih suka sesuatu yang pasti dengan hasil yang lebih rendah dari nilai moneter yang diharapkan dari suatu pertaruhan. Di sisi lain, di wilayah "putus asa" pada kekayaan negatif yang besar pada Gambar 7.2(b), perilaku tersebut adalah mencari risiko.

Nilai yang akan diterima agen sebagai ganti lotre disebut ekuivalen kepastian lotre. Penelitian telah menunjukkan bahwa kebanyakan orang akan menerima sekitar Rp 4.000.000 sebagai ganti pertaruhan yang memberikan Rp 10.000.000 setengah waktu dan Rp 0 setengah waktu lainnya—dengan kata lain, ekuivalen kepastian lotre adalah Rp 4.000.000, sedangkan EMV adalah Rp 5.000.000. Selisih antara EMV lotre dan ekuivalen kepastiannya disebut premi asuransi.

Penghindaran risiko adalah dasar bagi industri asuransi, karena itu berarti premi asuransi bersifat positif. Orang lebih suka membayar premi asuransi yang kecil daripada mempertaruhkan harga rumah mereka terhadap kemungkinan kebakaran. Dari sudut pandang perusahaan asuransi, harga rumah sangat kecil dibandingkan dengan total cadangan perusahaan. Ini berarti bahwa kurva utilitas perusahaan asuransi kira-kira linier pada wilayah yang sangat kecil, dan pertaruhan tersebut hampir tidak membebani perusahaan.

Perhatikan bahwa untuk perubahan kecil dalam kekayaan relatif terhadap kekayaan saat ini, hampir semua kurva akan kira-kira linier. Agen yang memiliki kurva linier dikatakan netral terhadap risiko. Oleh karena itu, untuk pertaruhan dengan jumlah kecil, kita mengharapkan netralitas risiko. Dalam arti tertentu, ini membenarkan prosedur sederhana yang mengusulkan pertaruhan kecil untuk menilai probabilitas dan membenarkan aksioma probabilitas.

Utilitas yang Diharapkan dan Kekecewaan Pascakeputusan

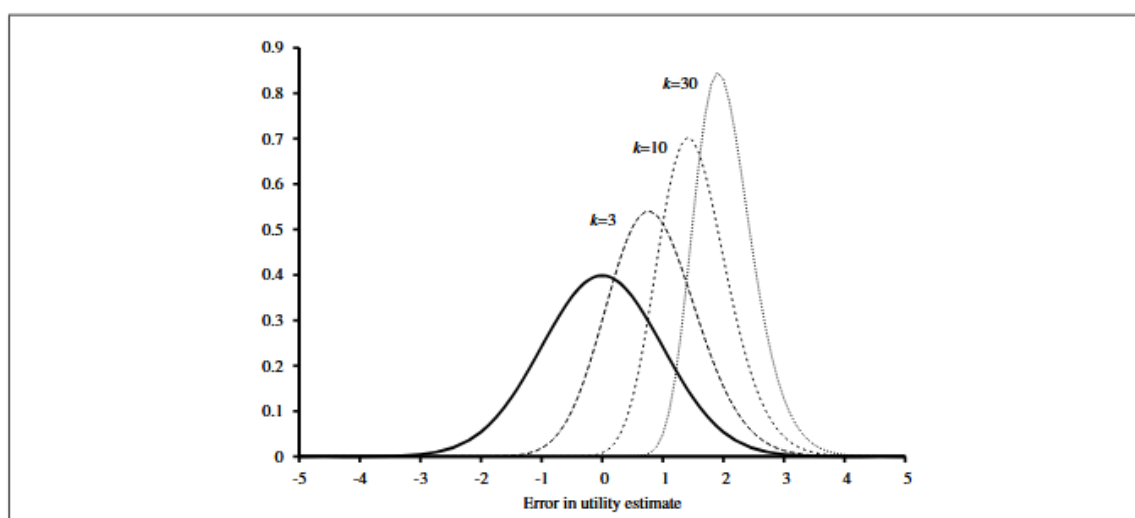
Cara rasional untuk memilih tindakan terbaik, a^* , adalah dengan memaksimalkan utilitas yang diharapkan:

$$a^* = \underset{a}{\operatorname{argmax}} EU(a | e)$$

Jika kita telah menghitung utilitas harapan dengan tepat menurut model probabilitas kita, dan jika model probabilitas tersebut dengan tepat mencerminkan proses stokastik yang mendasari

yang menghasilkan hasil, maka, secara rata-rata, kita akan mendapatkan utilitas yang kita harapkan jika seluruh proses diulang berkali-kali. Namun, pada kenyataannya, model kita biasanya terlalu menyederhanakan situasi sebenarnya, baik karena kita tidak cukup tahu (misalnya, ketika membuat keputusan investasi yang kompleks) atau karena perhitungan utilitas harapan yang sebenarnya terlalu sulit (misalnya, ketika memperkirakan utilitas status penerus simpul akar dalam permainan backgammon).

Dalam hal ini, kita sebenarnya bekerja dengan estimasi $\widehat{EU}(a | e)$ dari utilitas harapan yang sebenarnya. Kita akan berasumsi, mungkin dengan baik, bahwa estimasi tersebut tidak bias, yaitu, nilai harapan dari galat, $E(\widehat{EU}(a | e) - EU(a | e))$, adalah nol. Dalam hal ini, tampaknya masih masuk akal untuk memilih tindakan dengan estimasi utilitas tertinggi dan mengharapkan untuk menerima utilitas tersebut, secara rata-rata, ketika tindakan tersebut dieksekusi. Sayangnya, hasil sebenarnya biasanya akan jauh lebih buruk daripada yang kita perkirakan, meskipun perkiraannya tidak bias!



Gambar 7.3 Plot kesalahan pada masing-masing k estimasi utilitas dan distribusi maksimum k estimasi untuk $k = 3, 10$, dan 30 .

Untuk memahaminya, pertimbangkan masalah keputusan dengan k pilihan, yang masing-masing memiliki estimasi utilitas sebenarnya sebesar 0. Misalkan kesalahan dalam setiap estimasi utilitas memiliki rata-rata nol dan deviasi standar 1, ditunjukkan sebagai kurva tebal pada Gambar 7.3. Sekarang, saat kita benar-benar mulai menghasilkan estimasi, beberapa kesalahan akan negatif (pesimis) dan beberapa akan positif (optimis).

Karena kita memilih tindakan dengan estimasi utilitas tertinggi, kita jelas lebih menyukai estimasi yang terlalu optimis, dan itulah sumber biasnya. Menghitung distribusi nilai maksimum dari k estimasi dan dengan demikian mengkuantifikasi tingkat kekecewaan kita adalah hal yang mudah. Kurva pada Gambar 7.3 untuk $k = 3$ memiliki rata-rata sekitar 0,85, sehingga rata-rata kekecewaan akan menjadi sekitar 85% dari deviasi standar dalam estimasi utilitas. Dengan lebih banyak pilihan, estimasi yang sangat optimistis lebih mungkin muncul: untuk $k = 30$, kekecewaan akan sekitar dua kali lipat deviasi standar dalam estimasi.

Kecenderungan estimasi utilitas harapan dari pilihan terbaik yang terlalu tinggi ini disebut kutukan pengoptimal. Kutukan ini bahkan menimpa analis keputusan dan ahli statistik yang paling berpengalaman sekalipun. Manifestasi seriusnya termasuk meyakini bahwa obat baru yang menarik yang telah menyembuhkan 80% pasien dalam uji coba akan menyembuhkan 80% pasien (telah dipilih dari k = ribuan kandidat obat) atau bahwa reksa dana yang diiklankan memiliki imbal hasil di atas rata-rata akan terus memilikinya (telah dipilih untuk muncul dalam iklan dari k = puluhan dana dalam portofolio keseluruhan perusahaan). Bahkan bisa jadi apa yang tampak sebagai pilihan terbaik belum tentu demikian, jika varians dalam estimasi utilitasnya tinggi: suatu obat, yang dipilih dari ribuan pasien yang telah dicoba, yang telah menyembuhkan 9 dari 10 pasien kemungkinan lebih buruk daripada obat yang telah menyembuhkan 800 dari 1000 pasien.

Kutukan pengoptimal muncul di mana-mana karena proses seleksi yang memaksimalkan utilitas ada di mana-mana, sehingga menerima estimasi utilitas begitu saja adalah ide yang buruk. Kita dapat menghindari kutukan tersebut dengan menggunakan model probabilitas eksplisit $P(\widehat{EU} \mid EU)$ dari kesalahan dalam estimasi utilitas. Dengan model ini dan $P(EU)$ sebelumnya tentang apa yang mungkin kita harapkan secara wajar dari utilitas tersebut, kita memperlakukan estimasi utilitas, setelah diperoleh, sebagai bukti dan menghitung distribusi posterior untuk utilitas sebenarnya menggunakan aturan Bayes.

Penilaian dan Irasionalitas Manusia

Teori keputusan adalah teori normatif: teori ini menggambarkan bagaimana agen rasional seharusnya bertindak. Sebaliknya, teori deskriptif menggambarkan bagaimana agen yang sebenarnya—misalnya, manusia—sebenarnya bertindak. Penerapan teori ekonomi akan sangat ditingkatkan jika keduanya bertepatan, tetapi tampaknya ada beberapa bukti eksperimental yang bertentangan. Bukti tersebut menunjukkan bahwa manusia "dapat diprediksi tidak rasional".

Masalah yang paling terkenal adalah paradoks Allais. Orang-orang diberi pilihan antara lotere A dan B , lalu antara C dan D , yang masing-masing memiliki hadiah sebagai berikut:

A : 80% chance Rp 40.000.000

B : 100% chance Rp 30.000.000

C : 20% chance Rp 40.000.000

D : 25% chance Rp 30.000.000

Kebanyakan orang secara konsisten lebih memilih B daripada A (mengambil hal yang pasti), dan C daripada D (mengambil EMV yang lebih tinggi). Analisis normatif tidak setuju! Kita dapat melihat ini dengan lebih mudah jika kita menggunakan kebebasan yang tersirat oleh Persamaan (7.2) untuk menetapkan $U(Rp\ 0) = 0$. Dalam hal itu, maka $B \succ A$ menyiratkan bahwa $U(Rp\ 30.000.000) > 0,8 U(Rp\ 40.000.000)$, sedangkan $C \succ D$ menyiratkan sebaliknya. Dengan kata lain, tidak ada fungsi utilitas yang konsisten dengan pilihan-pilihan ini. Satu penjelasan untuk preferensi yang tampaknya tidak rasional adalah efek kepastian: orang sangat tertarik pada keuntungan yang pasti.

Ada beberapa alasan mengapa hal ini mungkin terjadi. Pertama, orang mungkin lebih suka mengurangi beban komputasi mereka; dengan memilih hasil tertentu, mereka tidak perlu

menghitung dengan probabilitas. Tetapi efeknya tetap ada bahkan ketika perhitungan yang terlibat adalah yang sangat mudah. Kedua, orang mungkin tidak mempercayai keabsahan probabilitas yang dinyatakan. Saya percaya bahwa lemparan koin kira-kira 50/50 jika saya memiliki kendali atas koin dan lemparannya, tetapi saya mungkin tidak mempercayai hasilnya jika lemparan dilakukan oleh seseorang dengan kepentingan pribadi dalam hasilnya.

Di hadapan ketidakpercayaan, mungkin lebih baik untuk memilih hal yang pasti. Ketiga, orang mungkin memperhitungkan keadaan emosional mereka serta keadaan keuangan mereka. Orang tahu bahwa mereka akan mengalami penyesalan jika mereka menyerahkan hadiah tertentu (B) untuk peluang 80% untuk mendapatkan hadiah yang lebih tinggi dan kemudian kalah. Dengan kata lain, jika A dipilih, ada peluang 20% untuk tidak mendapatkan uang dan merasa seperti orang bodoh, yang lebih buruk daripada tidak mendapatkan uang. Jadi mungkin orang yang memilih B daripada A dan C daripada D tidak bersikap tidak rasional; mereka hanya mengatakan bahwa mereka bersedia menyerahkan Rp 2.000.000 EMV untuk menghindari peluang 20% untuk merasa seperti orang bodoh.

Masalah terkait lainnya adalah paradoks Ellsberg. Di sini, hadiahnya tetap, tetapi probabilitasnya kurang dibatasi. Pembayaran Anda akan bergantung pada warna bola yang dipilih dari sebuah guci. Anda diberi tahu bahwa guci tersebut berisi $1/3$ bola merah, dan $2/3$ bola hitam atau kuning, tetapi Anda tidak tahu berapa banyak bola hitam dan berapa banyak bola kuning. Sekali lagi, Anda ditanya apakah Anda lebih suka lotere A atau B ; lalu C atau D :

A : Rp 1.000.000 for a red ball	C : Rp 1.000.000 for a red or yellow ball
B : Rp 1.000.000 for a black ball	D : Rp 1.000.000 for a black or yellow ball.

Seharusnya jelas bahwa jika Anda berpikir bola merah lebih banyak daripada bola hitam, maka Anda seharusnya lebih memilih A daripada B dan C daripada D ; jika Anda berpikir bola merah lebih sedikit daripada bola hitam, Anda seharusnya lebih memilih yang sebaliknya. Namun, ternyata kebanyakan orang lebih memilih A daripada B dan juga lebih memilih D daripada C , meskipun tidak ada keadaan dunia yang rasional untuk hal ini. Tampaknya orang-orang memiliki keengganan terhadap ambiguitas: A memberi Anda peluang $1/3$ untuk menang, sementara B bisa berada di antara 0 dan $2/3$. Demikian pula, D memberi Anda peluang $2/3$, sementara C bisa berada di antara $1/3$ dan $3/3$. Kebanyakan orang memilih probabilitas yang diketahui daripada yang tidak diketahui.

Masalah lainnya adalah bahwa rumusan yang tepat dari masalah keputusan dapat berdampak besar pada pilihan agen; ini disebut efek pembingkai. Eksperimen menunjukkan bahwa orang menyukai prosedur medis yang digambarkan memiliki "tingkat kelangsungan hidup 90%" sekitar dua kali lebih banyak daripada yang digambarkan memiliki "tingkat kematian 10%," meskipun kedua pernyataan ini memiliki arti yang persis sama. Perbedaan dalam penilaian ini telah ditemukan dalam berbagai eksperimen dan hampir sama apakah subjeknya adalah pasien di klinik, mahasiswa sekolah bisnis yang cangguh secara statistik, atau dokter yang berpengalaman.

Orang merasa lebih nyaman membuat penilaian utilitas relatif daripada yang absolut. Saya mungkin tidak tahu seberapa besar saya akan menikmati berbagai anggur yang ditawarkan oleh sebuah restoran. Restoran memanfaatkan ini dengan menawarkan sebotol anggur seharga Rp 2.000.000 yang mereka tahu tidak akan ada yang membeli, tetapi berfungsi untuk mendistorsi perkiraan pelanggan tentang nilai semua anggur dan membuat sebotol anggur seharga Rp 550.000 tampak seperti barang murah. Ini disebut efek penjangkaran.

Jika informan manusia bersikeras pada penilaian preferensi yang kontradiktif, tidak ada yang dapat dilakukan agen otomatis untuk konsisten dengan penilaian tersebut. Untungnya, penilaian preferensi yang dibuat oleh manusia sering kali terbuka untuk direvisi berdasarkan pertimbangan lebih lanjut. Paradoks seperti paradoks Allais sangat berkurang jika pilihan dijelaskan dengan lebih baik. Dalam penelitian di Harvard Business School tentang penilaian utilitas uang, Keeney dan Raiffa menemukan hal berikut:

Subjek cenderung terlalu menghindari risiko dalam fungsi utilitas yang kecil dan karenanya ... menunjukkan premi risiko yang sangat besar untuk lotere dengan spread yang besar. Namun, sebagian besar subjek dapat mendamaikan ketidakkonsistenan mereka dan merasa bahwa mereka telah mempelajari pelajaran penting tentang bagaimana mereka ingin berperilaku. Akibatnya, beberapa subjek membatalkan asuransi tabrakan mobil mereka dan mengambil lebih banyak asuransi berjangka untuk jiwa mereka.

Bukti irasionalitas manusia juga dipertanyakan oleh para peneliti di bidang psikologi evolusi, yang menunjukkan fakta bahwa mekanisme pengambilan keputusan otak kita tidak berevolusi untuk memecahkan masalah kata dengan probabilitas dan hadiah yang dinyatakan sebagai angka desimal. Mari kita akui, demi argumen, bahwa otak memiliki mekanisme saraf bawaan untuk menghitung dengan probabilitas dan utilitas, atau sesuatu yang secara fungsional setara; jika demikian, masukan yang diperlukan akan diperoleh melalui pengalaman hasil dan penghargaan yang terakumulasi daripada melalui presentasi linguistik nilai numerik.

Jauh dari jelas bahwa kita dapat langsung mengakses mekanisme saraf bawaan otak dengan menyajikan masalah keputusan dalam bentuk linguistik/numerik. Fakta bahwa kata-kata yang berbeda dari masalah keputusan yang sama menimbulkan pilihan yang berbeda menunjukkan bahwa masalah keputusan itu sendiri tidak berhasil. Didorong oleh pengamatan ini, para psikolog telah mencoba menyajikan masalah dalam penalaran yang tidak pasti dan pengambilan keputusan dalam bentuk yang "tepat secara evolusi"; misalnya, alih-alih mengatakan "tingkat kelangsungan hidup 90%," peneliti mungkin menunjukkan 100 animasi figur tongkat operasi, di mana pasien meninggal dalam 10 dari animasi tersebut dan bertahan hidup dalam 90 animasi. (Kebosanan merupakan faktor yang mempersulit percobaan ini!) Dengan masalah keputusan yang diajukan dengan cara ini, orang-orang tampaknya jauh lebih dekat dengan perilaku rasional daripada yang diduga sebelumnya.

7.4 FUNGSI UTILITAS MULTIATTRIBUT

Pengambilan keputusan di bidang kebijakan publik melibatkan taruhan tinggi, baik dalam bentuk uang maupun nyawa. Misalnya, dalam memutuskan tingkat emisi berbahaya yang diizinkan dari pembangkit listrik, pembuat kebijakan harus mempertimbangkan pencegahan kematian dan kecacatan dengan manfaat listrik dan beban ekonomi dari mitigasi

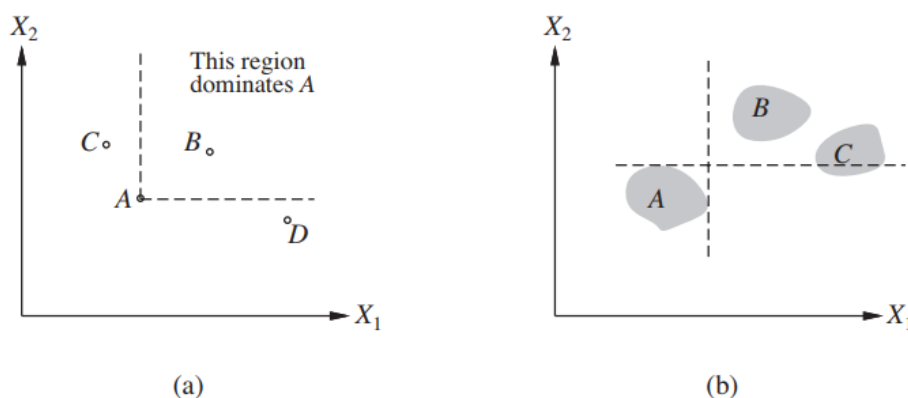
emisi. Penempatan bandara baru memerlukan pertimbangan gangguan yang disebabkan oleh konstruksi; biaya lahan; jarak dari pusat populasi; kebisingan operasi penerbangan; masalah keselamatan yang timbul dari topografi lokal dan kondisi cuaca; dan sebagainya. Masalah seperti ini, yang hasilnya dicirikan oleh dua atau lebih atribut, ditangani oleh teori utilitas multiatribut.

Kita akan menyebut atribut $X = X_1, \dots, X_n$; vektor penugasan yang lengkap adalah $x = \langle x_1, \dots, x_n \rangle$, di mana setiap x_i berupa nilai numerik atau nilai diskrit dengan urutan nilai yang diasumsikan. Kita akan berasumsi bahwa nilai atribut yang lebih tinggi bersesuaian dengan utilitas yang lebih tinggi, dengan asumsi faktor-faktor lain tetap sama. Misalnya, jika kita memilih *AbsenceOfNoise* sebagai atribut dalam permasalahan bandara, semakin besar nilainya, semakin baik solusinya. Kita mulai dengan memeriksa kasus-kasus di mana keputusan dapat dibuat tanpa menggabungkan nilai-nilai atribut menjadi satu nilai utilitas. Kemudian, kita melihat kasus-kasus di mana utilitas kombinasi atribut dapat dispesifikasikan secara ringkas.

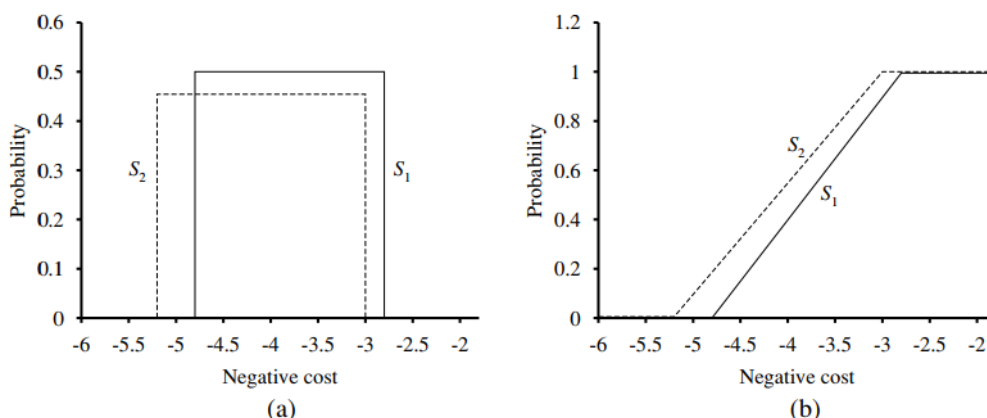
Dominasi

Misalkan lokasi bandara S_1 berbiaya lebih rendah, menghasilkan polusi suara yang lebih sedikit, dan lebih aman daripada lokasi S_2 . Kita tidak akan ragu untuk menolak S_2 . Kemudian, kita katakan bahwa terdapat dominasi ketat S_1 atas S_2 . Secara umum, jika suatu opsi memiliki nilai yang lebih rendah pada semua atribut dibandingkan opsi lain, opsi tersebut tidak perlu dipertimbangkan lebih lanjut.

Dominasi ketat seringkali sangat berguna dalam mempersempit pilihan menjadi pesaing yang sebenarnya, meskipun jarang menghasilkan pilihan yang unik. Gambar 7.4(a) menunjukkan diagram skematik untuk kasus dua atribut. Hal ini berlaku untuk kasus deterministik, di mana nilai atributnya diketahui secara pasti. Bagaimana dengan kasus umum, di mana hasilnya tidak pasti? Analogi langsung dari dominasi ketat dapat dibangun, di mana, terlepas dari ketidakpastiannya, semua kemungkinan hasil konkret untuk S_1 secara ketat mendominasi semua kemungkinan hasil untuk S_2 . Tentu saja, hal ini mungkin akan terjadi lebih jarang daripada dalam kasus deterministik.



Gambar 7.4 Dominasi ketat. (a) Deterministik: Pilihan A didominasi secara ketat oleh B, tetapi tidak oleh C atau D. (b) Tidak pasti: A didominasi secara ketat oleh B, tetapi tidak oleh C.



Gambar 7.5 Dominasi stokastik. (a) S_1 secara stokastik mendominasi S_2 berdasarkan biaya. (b) Distribusi kumulatif untuk biaya negatif S_1 dan S_2 .

Untungnya, ada generalisasi yang lebih bermanfaat yang disebut dominasi stokastik, yang sangat sering terjadi dalam permasalahan nyata. Dominasi stokastik paling mudah dipahami dalam konteks satu atribut. Misalkan kita meyakini bahwa biaya penempatan bandara di S_1 terdistribusi secara seragam antara Rp 28 triliun dan Rp 48 triliun dan bahwa biaya di S_2 terdistribusi secara seragam antara Rp 30 triliun dan Rp 52 triliun. Gambar 7.5(a) menunjukkan distribusi ini, dengan biaya diplot sebagai nilai negatif.

Kemudian, hanya dengan informasi bahwa utilitas menurun seiring dengan biaya, kita dapat mengatakan bahwa S_1 secara stokastik mendominasi S_2 (yaitu, S_2 dapat diabaikan). Penting untuk dicatat bahwa hal ini tidak mengikuti perbandingan biaya yang diharapkan. Misalnya, jika kita mengetahui biaya S_1 tepat Rp 38 triliun, maka kita tidak akan dapat membuat keputusan tanpa informasi tambahan tentang utilitas uang. (Mungkin tampak aneh bahwa informasi lebih lanjut tentang biaya S_1 dapat membuat agen kurang mampu memutuskan. Paradoks ini diselesaikan dengan mencatat bahwa tanpa adanya informasi biaya yang pasti, keputusan lebih mudah dibuat tetapi lebih mungkin salah.)

Hubungan pasti antara distribusi atribut yang diperlukan untuk menetapkan dominasi stokastik paling baik dilihat dengan memeriksa distribusi kumulatif, yang ditunjukkan pada Gambar 7.5(b). Distribusi kumulatif mengukur probabilitas bahwa biaya kurang dari atau sama dengan jumlah tertentu—yaitu, mengintegrasikan distribusi asli. Jika distribusi kumulatif untuk S_1 selalu berada di sebelah kanan distribusi kumulatif untuk S_2 , maka, secara stokastik, S_1 lebih murah daripada S_2 . Secara formal, jika dua tindakan A_1 dan A_2 menghasilkan distribusi probabilitas $p_1(x)$ dan $p_2(x)$ pada atribut X , maka A_1 secara stokastik mendominasi A_2 pada X jika,

$$\forall_x \int_{-\infty}^x p_1(x') dx' \leq \int_{-\infty}^x p_2(x') dx' .$$

Relevansi definisi ini terhadap pemilihan keputusan optimal berasal dari sifat berikut: jika A_1 secara stokastik mendominasi A_2 , maka untuk setiap fungsi utilitas monotonik non-menurun

$U(x)$, utilitas ekspektasi A_1 setidaknya sama tingginya dengan utilitas ekspektasi A_2 . Oleh karena itu, jika suatu tindakan didominasi secara stokastik oleh tindakan lain pada semua atribut, maka tindakan tersebut dapat diabaikan.

Kondisi dominasi stokastik mungkin tampak agak teknis dan mungkin tidak mudah dievaluasi tanpa perhitungan probabilitas yang ekstensif. Faktanya, kondisi ini dapat diputuskan dengan sangat mudah dalam banyak kasus. Misalkan, biaya transportasi konstruksi bergantung pada jarak ke pemasok. Biaya itu sendiri tidak pasti, tetapi semakin jauh jaraknya, semakin besar pula biayanya.

Jika S_1 lebih dekat daripada S_2 , maka S_1 akan mendominasi S_2 dalam hal biaya. Meskipun kami tidak akan menyajikannya di sini, terdapat algoritma untuk menyebarkan informasi kualitatif semacam ini di antara variabel-variabel yang tidak pasti dalam jaringan probabilistik kualitatif, yang memungkinkan sistem untuk membuat keputusan rasional berdasarkan dominasi stokastik, tanpa menggunakan nilai numerik apa pun.

Struktur Preferensi dan Utilitas Multiatribut

Misalkan kita memiliki n atribut, yang masing-masing memiliki d kemungkinan nilai yang berbeda. Untuk menentukan fungsi utilitas lengkap $U(x_1, \dots, x_n)$, kita membutuhkan d^n nilai dalam kasus terburuk. Kasus terburuk berkaitan dengan situasi di mana preferensi agen tidak memiliki keteraturan sama sekali. Teori utilitas multiatribut didasarkan pada anggapan bahwa preferensi agen tipikal memiliki struktur yang jauh lebih kompleks. Pendekatan dasarnya adalah mengidentifikasi keteraturan dalam perilaku preferensi yang kita harapkan dan menggunakan apa yang disebut teorema representasi untuk menunjukkan bahwa agen dengan jenis struktur preferensi tertentu memiliki fungsi utilitas.

$$U(x_1, \dots, x_n) = F[f_1(x_1), \dots, f_n(x_n)],$$

Di mana F , kami harap, adalah fungsi sederhana seperti penjumlahan. Perhatikan kemiripan dengan penggunaan jaringan Bayesian untuk menguraikan probabilitas gabungan dari beberapa variabel acak.

Preferensi Tanpa Ketidakpastian

Mari kita mulai dengan kasus deterministik. Ingat bahwa untuk lingkungan deterministik agen memiliki fungsi nilai $V(x_1, \dots, x_n)$; tujuannya adalah untuk merepresentasikan fungsi ini secara ringkas. Keteraturan dasar yang muncul dalam struktur preferensi deterministik disebut independensi preferensi. Dua atribut X_1 dan X_2 secara preferensial independen dari atribut ketiga X_3 jika preferensi antara hasil $\langle x_1, x_2, x_3 \rangle$ dan $\langle x_1^1, x_2^1, x_3 \rangle$ tidak bergantung pada nilai tertentu x_3 untuk atribut X_3 .

Kembali ke contoh bandara, di mana kita memiliki (di antara atribut lainnya) *Noise*, *Cost*, dan *Deaths* untuk dipertimbangkan, seseorang dapat mengusulkan bahwa *Noise* dan *Cost* secara preferensial independen dari *Deaths*. Misalnya, jika kita lebih memilih negara bagian dengan 20.000 orang yang tinggal di jalur penerbangan dan biaya konstruksi sebesar Rp 40 triliun daripada negara bagian dengan 70.000 orang yang tinggal di jalur penerbangan dan biaya sebesar Rp 37 triliun ketika tingkat keselamatan adalah 0,06 kematian

per juta mil penumpang dalam kedua kasus, maka kita akan memiliki preferensi yang sama ketika tingkat keselamatan adalah 0,12 atau 0,03; dan independensi yang sama akan berlaku untuk preferensi antara pasangan nilai lainnya untuk Kebisingan dan Biaya.

Jelas juga bahwa *Cost* dan *Deaths* lebih disukai independen dari *Noise* dan bahwa *Noise* dan *Deaths* lebih disukai independen dari Biaya. Kita katakan bahwa kumpulan atribut $\{Noise, Cost, Deaths\}$ menunjukkan independensi preferensial bersama (MPI). MPI mengatakan bahwa, meskipun setiap atribut mungkin penting, itu tidak mempengaruhi cara seseorang memperdagangkan atribut lainnya satu sama lain. Kemandirian preferensial bersama merupakan sesuatu yang sulit diucapkan, namun berkat teorema luar biasa yang dikemukakan oleh ekonom Ge'rrard Debreu, kita dapat memperoleh bentuk yang sangat sederhana untuk fungsi nilai agen: Jika atribut $X_1, \dots, X_n$ saling bebas preferensial, maka perilaku preferensi agen dapat digambarkan sebagai memaksimalkan fungsi

$$V(x_1, \dots, x_n) = \sum_i V_i(x_i),$$

di mana setiap V_i adalah fungsi nilai yang hanya merujuk pada atribut X_i . Misalnya, keputusan bandara mungkin dapat dibuat menggunakan fungsi nilai

$$V(Noise, Cost, Deaths) = -Noise \times 10^4 - Cost - Deaths \times 10^{12}.$$

Fungsi nilai jenis ini disebut fungsi nilai aditif. Fungsi aditif adalah cara yang sangat alami untuk menggambarkan preferensi agen dan valid dalam banyak situasi dunia nyata. Untuk n atribut, menilai fungsi nilai aditif memerlukan penilaian n fungsi nilai satu dimensi yang terpisah, alih-alih satu fungsi n -dimensi; biasanya, ini merepresentasikan pengurangan eksponensial dalam jumlah eksperimen preferensi yang diperlukan.

Bahkan ketika MPI tidak sepenuhnya berlaku, seperti yang mungkin terjadi pada nilai ekstrem atribut, fungsi nilai aditif mungkin masih memberikan perkiraan yang baik untuk preferensi agen. Hal ini terutama berlaku ketika pelanggaran MPI terjadi di bagian rentang atribut yang tidak mungkin terjadi dalam praktik. Untuk memahami MPI dengan lebih baik, ada baiknya melihat kasus-kasus di mana ia tidak berlaku. Misalkan Anda berada di pasar abad pertengahan, mempertimbangkan pembelian beberapa anjing pemburu, beberapa ayam, dan beberapa kandang anyaman untuk ayam. Anjing pemburu sangat berharga, tetapi jika Anda tidak memiliki cukup kandang untuk ayam, anjing akan memakan ayam; Oleh karena itu, tradeoff antara anjing dan ayam sangat bergantung pada jumlah kandang, dan MPI dilanggar. Adanya interaksi semacam ini di antara berbagai atribut membuat penilaian fungsi nilai secara keseluruhan menjadi jauh lebih sulit.

Preferensi dengan Ketidakpastian

Ketika ketidakpastian hadir dalam domain, kita juga perlu mempertimbangkan struktur preferensi antar lotere dan memahami sifat-sifat fungsi utilitas yang dihasilkan, alih-alih hanya fungsi nilai. Matematika dari masalah ini bisa menjadi sangat rumit, jadi kami hanya

menyajikan salah satu hasil utama untuk memberikan gambaran tentang apa yang bisa dilakukan. Pembaca dapat merujuk ke Keeney dan Raiffa untuk survei menyeluruh di bidang ini.

Gagasan dasar independensi utilitas memperluas independensi preferensi untuk mencakup lotere: satu set atribut X bersifat independen utilitas terhadap satu set atribut Y jika preferensi antar lotere pada atribut-atribut di X bersifat independen terhadap nilai-nilai atribut tertentu di Y . Satu set atribut bersifat saling independen utilitas (MUI) jika setiap subsetnya bersifat independen utilitas terhadap atribut-atribut yang tersisa. Sekali lagi, tampaknya masuk akal untuk mengusulkan bahwa atribut-atribut bandara bersifat MUI.

MUI menyiratkan bahwa perilaku agen dapat dijelaskan menggunakan fungsi utilitas perkalian. Bentuk umum fungsi utilitas perkalian paling baik dilihat dengan melihat kasus untuk tiga atribut. Untuk lebih ringkas, kami menggunakan U_i untuk menggantikan $U_i(x_i)$:

$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + k_1k_2k_3U_1U_2U_3 .$$

Meskipun terlihat tidak sederhana, model ini hanya berisi tiga fungsi utilitas atribut tunggal dan tiga konstanta. Secara umum, masalah n -atribut yang menunjukkan MUI dapat dimodelkan menggunakan n utilitas atribut tunggal dan n konstanta. Masing-masing fungsi utilitas atribut tunggal dapat dikembangkan secara independen dari atribut lainnya, dan kombinasi ini dijamin menghasilkan preferensi keseluruhan yang tepat. Asumsi tambahan diperlukan untuk mendapatkan fungsi utilitas yang murni aditif.

7.5 JARINGAN KEPUTUSAN

Di bagian ini, kita akan membahas mekanisme umum untuk membuat keputusan rasional. Notasi ini sering disebut diagram pengaruh, tetapi kita akan menggunakan istilah yang lebih deskriptif, yaitu jaringan keputusan. Jaringan keputusan menggabungkan jaringan Bayesian dengan jenis simpul tambahan untuk tindakan dan utilitas. Kita menggunakan penentuan lokasi bandara sebagai contoh.

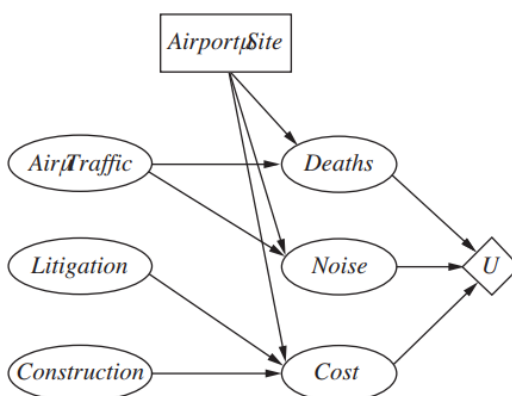
Merepresentasikan Masalah Keputusan dengan Jaringan Keputusan

Dalam bentuk paling umum, jaringan keputusan merepresentasikan informasi tentang status agen saat ini, kemungkinan tindakannya, status yang akan dihasilkan dari tindakan agen, dan utilitas status tersebut. Oleh karena itu, jaringan ini menyediakan substrat untuk mengimplementasikan agen berbasis utilitas. Gambar 7.6 menunjukkan jaringan keputusan untuk masalah penentuan lokasi bandara. Gambar ini mengilustrasikan tiga jenis simpul yang digunakan:

- Simpul peluang (oval) merepresentasikan variabel acak, sebagaimana dalam jaringan Bayesian. Agen mungkin tidak yakin tentang biaya konstruksi, tingkat lalu lintas udara, dan potensi litigasi, serta variabel Kematian, Kebisingan, dan Total Biaya, yang masing-masing juga bergantung pada lokasi yang dipilih. Setiap simpul peluang memiliki distribusi kondisional yang terkait dengannya, yang diindeks oleh status simpul induk. Dalam jaringan keputusan, simpul induk dapat mencakup simpul keputusan serta

simpul peluang. Perhatikan bahwa setiap simpul peluang status saat ini dapat menjadi bagian dari jaringan Bayesian yang besar untuk menilai biaya konstruksi, tingkat lalu lintas udara, atau potensi litigasi.

- Simpul keputusan (persegi panjang) merepresentasikan titik-titik di mana pembuat keputusan memiliki pilihan tindakan. Dalam hal ini, tindakan *AirportSite* dapat memiliki nilai yang berbeda untuk setiap lokasi yang dipertimbangkan. Pilihan tersebut memengaruhi biaya, keselamatan, dan kebisingan yang akan dihasilkan. Dalam bab ini, kami berasumsi bahwa kami berurusan dengan satu simpul keputusan.
- Simpul utilitas (berlian) merepresentasikan fungsi utilitas agen. Simpul utilitas memiliki induk yang terdiri dari semua variabel yang menggambarkan hasil yang secara langsung memengaruhi utilitas. Simpul utilitas dikaitkan dengan deskripsi utilitas agen sebagai fungsi dari atribut induk. Deskripsi tersebut dapat berupa tabulasi fungsi, atau dapat berupa fungsi aditif atau linear berparameter dari nilai-nilai atribut.

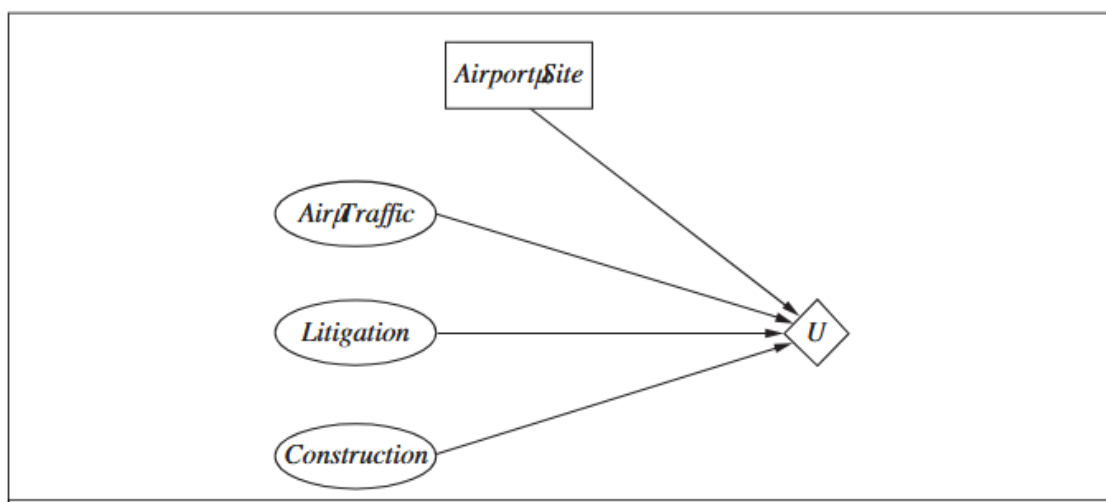


Gambar 7.6 Jaringan keputusan sederhana untuk masalah penentuan lokasi bandara.

Bentuk yang disederhanakan juga digunakan dalam banyak kasus. Notasinya tetap identik, tetapi simpul peluang yang menggambarkan status hasil dihilangkan. Sebagai gantinya, simpul utilitas terhubung langsung ke simpul status saat ini dan simpul keputusan. Dalam hal ini, alih-alih merepresentasikan fungsi utilitas pada status hasil, simpul utilitas merepresentasikan utilitas yang diharapkan terkait dengan setiap tindakan, sebagaimana didefinisikan dalam Persamaan (7.1); artinya, simpul tersebut dikaitkan dengan fungsi utilitas tindakan (juga dikenal sebagai fungsi Q dalam pembelajaran penguatan. Gambar 7.7 menunjukkan representasi utilitas tindakan dari masalah penentuan lokasi bandara.

Perhatikan bahwa, karena simpul peluang Kebisingan, Kematian, dan Biaya pada Gambar 7.6 merujuk pada status mendatang, nilai-nilai tersebut tidak boleh ditetapkan sebagai variabel bukti. Dengan demikian, versi yang disederhanakan yang menghilangkan simpul-simpul ini dapat digunakan kapan pun bentuk yang lebih umum dapat digunakan. Meskipun bentuk yang disederhanakan berisi lebih sedikit simpul, penghilangan deskripsi eksplisit tentang hasil keputusan lokasi menyebabkannya kurang fleksibel terhadap perubahan keadaan.

Misalnya, pada Gambar 7.6, perubahan tingkat kebisingan pesawat dapat tercermin melalui perubahan tabel probabilitas bersyarat yang terkait dengan simpul Kebisingan, sedangkan perubahan bobot yang diberikan pada polusi kebisingan dalam fungsi utilitas dapat tercermin melalui perubahan tabel utilitas. Di sisi lain, pada diagram tindakan-utilitas, Gambar 7.7, semua perubahan tersebut harus tercermin melalui perubahan pada tabel tindakan-utilitas. Pada dasarnya, formulasi tindakan-utilitas merupakan versi kompilasi dari formulasi aslinya.



Gambar 7.7 Representasi sederhana dari masalah penentuan lokasi bandara. Node peluang yang sesuai dengan status hasil telah difaktorkan.

Mengevaluasi Jaringan Keputusan

Tindakan dipilih dengan mengevaluasi jaringan keputusan untuk setiap kemungkinan pengaturan simpul keputusan. Setelah simpul keputusan ditetapkan, ia berperilaku persis seperti simpul peluang yang telah ditetapkan sebagai variabel bukti. Algoritme untuk mengevaluasi jaringan keputusan adalah sebagai berikut:

1. Tetapkan variabel bukti untuk kondisi saat ini.
2. Untuk setiap kemungkinan nilai simpul keputusan:
 - (a) Tetapkan simpul keputusan ke nilai tersebut.
 - (b) Hitung probabilitas posterior untuk simpul induk dari simpul utilitas, menggunakan algoritma inferensi probabilistik standar.
 - (c) Hitung utilitas yang dihasilkan untuk tindakan tersebut.
3. Kembalikan tindakan dengan utilitas tertinggi.

Ini merupakan perluasan langsung dari algoritma jaringan Bayesian dan dapat langsung diintegrasikan ke dalam desain agen. Kita akan melihat di Bab 17 bahwa kemungkinan mengeksekusi beberapa tindakan secara berurutan membuat permasalahan ini jauh lebih menarik.

7.6 NILAI INFORMASI

Dalam analisis sebelumnya, kami berasumsi bahwa semua informasi yang relevan, atau setidaknya semua informasi yang tersedia, diberikan kepada agen sebelum ia membuat keputusan. Dalam praktiknya, hal ini hampir tidak pernah terjadi. Salah satu bagian terpenting dari pengambilan keputusan adalah mengetahui pertanyaan apa yang harus diajukan. Misalnya, seorang dokter tidak dapat berharap untuk diberikan hasil dari semua tes diagnostik dan pertanyaan yang mungkin diajukan saat pasien pertama kali memasuki ruang konsultasi. Tes seringkali mahal dan terkadang berbahaya. Pentingnya tes tersebut bergantung pada dua faktor: apakah hasil tes akan menghasilkan rencana perawatan yang jauh lebih baik, dan seberapa besar kemungkinan berbagai hasil tes tersebut.

Bagian ini menjelaskan teori nilai informasi, yang memungkinkan agen untuk memilih informasi apa yang akan diperoleh. Kami berasumsi bahwa, sebelum memilih tindakan "nyata" yang direpresentasikan oleh simpul keputusan, agen dapat memperoleh nilai dari setiap variabel peluang yang berpotensi teramati dalam model. Dengan demikian, teori nilai informasi melibatkan bentuk pengambilan keputusan sekuensial yang disederhanakan—disederhanakan karena tindakan observasi hanya memengaruhi kondisi keyakinan agen, bukan kondisi fisik eksternal. Nilai setiap observasi tertentu harus berasal dari potensi untuk memengaruhi tindakan fisik agen pada akhirnya; dan potensi ini dapat diperkirakan langsung dari model keputusan itu sendiri.

Contoh Sederhana

Misalkan sebuah perusahaan minyak berharap untuk membeli satu dari n blok hak pengeboran laut yang tidak dapat dibedakan. Mari kita asumsikan lebih lanjut bahwa tepat satu blok mengandung minyak senilai C rupiah, sementara yang lain tidak bernilai. Harga yang diminta untuk setiap blok adalah C/n rupiah. Jika perusahaan netral terhadap risiko, maka perusahaan akan bersikap acuh tak acuh antara membeli satu blok dan tidak membelinya.

Sekarang, misalkan seorang seismolog menawarkan kepada perusahaan hasil survei blok nomor 3, yang menunjukkan secara pasti apakah blok tersebut mengandung minyak. Berapa banyak yang seharusnya bersedia dibayarkan perusahaan untuk informasi tersebut? Cara menjawab pertanyaan ini adalah dengan memeriksa apa yang akan dilakukan perusahaan jika memiliki informasi berikut:

- Dengan probabilitas $1/n$, survei akan menunjukkan adanya minyak di blok 3. Dalam hal ini, perusahaan akan membeli blok 3 seharga C/n rupiah dan memperoleh laba sebesar $C - C/n = (n - 1) C/n$ rupiah.
- Dengan probabilitas $(n-1)/n$, survei akan menunjukkan bahwa blok tersebut tidak mengandung minyak, sehingga perusahaan akan membeli blok yang berbeda. Probabilitas menemukan minyak di salah satu blok lainnya berubah dari $1/n$ menjadi $1/(n - 1)$, sehingga perusahaan memperoleh laba yang diharapkan sebesar $C/(n - 1) - C/n = C/n(n - 1)$ rupiah.

Sekarang, kita dapat menghitung laba yang diharapkan, dengan informasi survei berikut:

$$\frac{1}{n}x \frac{(n-1)C}{n} + \frac{n-1}{n}x \frac{C}{n(n-1)} = C/n .$$

Oleh karena itu, perusahaan harus bersedia membayar seismolog hingga C/n rupiah untuk informasi tersebut: informasi tersebut bernilai sama dengan blok itu sendiri.

Nilai informasi berasal dari fakta bahwa dengan informasi tersebut, tindakan seseorang dapat diubah agar sesuai dengan situasi yang sebenarnya. Seseorang dapat melakukan diskriminasi berdasarkan situasi, sedangkan tanpa informasi, seseorang harus melakukan yang terbaik secara rata-rata atas situasi yang mungkin terjadi. Secara umum, nilai dari informasi tertentu didefinisikan sebagai perbedaan nilai yang diharapkan antara tindakan terbaik sebelum dan sesudah informasi diperoleh.

Rumus Umum untuk Informasi Sempurna

Mudah untuk menurunkan rumus matematika umum untuk nilai informasi. Kita berasumsi bahwa bukti eksak dapat diperoleh mengenai nilai suatu variabel acak E_j (yaitu, kita mengetahui $E_j = e_j$), sehingga frasa nilai informasi sempurna (VPI) digunakan.

Misalkan bukti awal agen adalah e . Maka nilai tindakan terbaik saat ini, α , didefinisikan oleh

$$EU(\alpha | e) = \max_a \sum_{s'} P(\text{HASIL}(a) = s' | a, e) U(s'),$$

dan nilai tindakan terbaik baru (setelah bukti baru $E_j = e_j$ diperoleh) akan menjadi

$$EU(a_{e_j} | e, e_j) = \max_a \sum_{s'} P(\text{HASIL}(a) = s' | a, e, e_j) U(s'),$$

Namun E_j merupakan variabel acak yang nilainya saat ini tidak diketahui, jadi untuk menentukan nilai penemuan E_j , dengan informasi terkini e , kita harus menghitung rata-rata semua nilai e_{jk} yang mungkin kita temukan untuk E_j , dengan menggunakan keyakinan kita saat ini mengenai nilainya:

$$VPI_e(E_j) = \left(\sum_k P(E_j = e_{jk} | e) EU(a_{e_{jk}} | e, E_j = e_{jk}) \right) - EU(\alpha | e) .$$

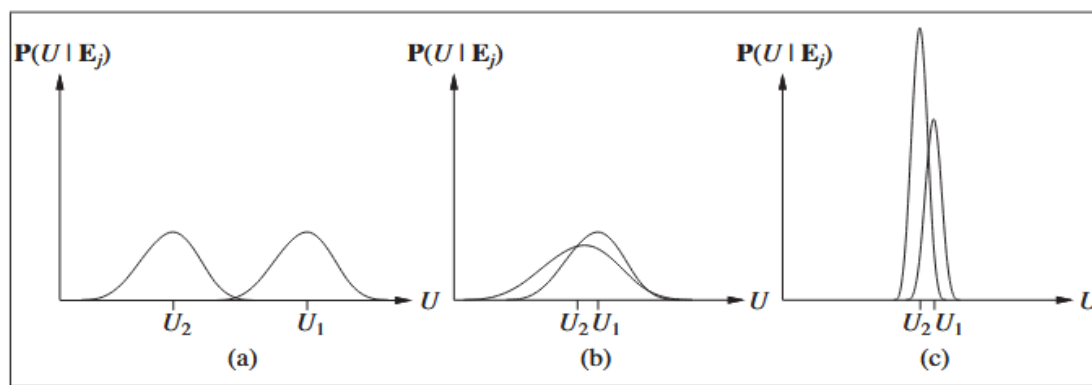
Untuk mendapatkan intuisi mengenai rumus ini, pertimbangkan kasus sederhana di mana hanya ada dua tindakan, a_1 dan a_2 , yang dapat dipilih. Utilitas yang diharapkan saat ini adalah U_1 dan U_2 . Informasi $E_j = e_{jk}$ akan menghasilkan beberapa utilitas baru yang diharapkan U_1^i dan U_2^i untuk tindakan tersebut, tetapi sebelum kita memperoleh E_j , kita akan memiliki beberapa distribusi probabilitas atas nilai-nilai yang mungkin dari U_1^i dan U_2^i (yang kita asumsikan bersifat independen).

Misalkan a_1 dan a_2 mewakili dua rute berbeda melalui pegunungan di musim dingin. a_1 adalah jalan raya lurus yang bagus melalui jalan setapak yang rendah, dan a_2 adalah jalan tanah yang berkelok-kelok di atasnya. Berdasarkan informasi ini, a_1 jelas lebih disukai, karena sangat mungkin a_2 terhalang oleh longsor salju, sedangkan tidak mungkin ada sesuatu yang menghalangi a_1 . Oleh karena itu, U_1 jelas lebih tinggi daripada U_2 . Dimungkinkan untuk memperoleh laporan satelit E_j tentang kondisi aktual setiap jalan yang akan memberikan harapan baru, U_1^1 dan U_2^1 , untuk dua penyeberangan tersebut. Distribusi untuk ekspektasi ini ditunjukkan pada Gambar 7.8(a). Jelas, dalam kasus ini, tidak ada gunanya mengeluarkan biaya untuk memperoleh laporan satelit, karena kecil kemungkinan informasi yang diperoleh dari laporan tersebut akan mengubah rencana. Tanpa perubahan, informasi tidak memiliki nilai.

Sekarang anggaplah kita memilih antara dua jalan tanah berkelok yang berbeda dengan panjang yang sedikit berbeda dan kita membawa penumpang yang terluka parah. Kemudian, bahkan ketika U_1 dan U_2 cukup dekat, distribusi U_1^1 dan U_2^1 sangat luas. Ada kemungkinan yang signifikan bahwa rute kedua akan menjadi bebas sementara yang pertama terhalang, dan dalam kasus ini perbedaan utilitas akan sangat tinggi. Rumus VPI menunjukkan bahwa mungkin ada baiknya memperoleh laporan satelit. Situasi seperti itu ditunjukkan pada Gambar 7.8(b).

Terakhir, anggaplah kita memilih antara dua jalan tanah di musim panas, ketika penyumbatan oleh longsor salju tidak mungkin terjadi. Dalam kasus ini, laporan satelit mungkin menunjukkan satu rute lebih indah daripada yang lain karena padang rumput Alpen yang berbunga, atau mungkin lebih basah karena aliran sungai yang menyimpang. Oleh karena itu, sangat mungkin kami akan mengubah rencana kami jika kami memiliki informasi tersebut. Namun, dalam kasus ini, perbedaan nilai antara kedua rute tersebut kemungkinan masih sangat kecil, jadi kami tidak akan repot-repot untuk mendapatkan laporan. Situasi ini ditunjukkan pada Gambar 7.8(c).

Singkatnya, informasi memiliki nilai sejauh informasi tersebut kemungkinan dapat menyebabkan perubahan rencana dan sejauh rencana baru tersebut akan jauh lebih baik daripada rencana lama.



Gambar 7.8 Tiga kasus umum untuk nilai informasi. Dalam (a), a_1 hampir pasti akan tetap lebih unggul daripada a_2 , jadi informasinya tidak diperlukan. Dalam (b), pilihannya tidak jelas dan informasinya penting. Dalam (c), pilihannya tidak jelas, tetapi karena tidak banyak

berpengaruh, informasinya kurang berharga. (Catatan: Fakta bahwa U_2 memiliki puncak yang tinggi dalam (c) berarti bahwa nilai yang diharapkan diketahui dengan kepastian yang lebih tinggi daripada U_1 .)

Sifat-sifat Nilai Informasi

Orang mungkin bertanya apakah mungkin suatu informasi bersifat merugikan: dapatkah informasi tersebut benar-benar memiliki nilai harapan negatif? Secara intuitif, hal ini seharusnya mustahil. Lagipula, dalam kasus terburuk, seseorang dapat mengabaikan informasi tersebut dan berpura-pura tidak pernah menerimanya. Hal ini dikonfirmasi oleh teorema berikut, yang berlaku untuk semua agen teoretis-keputusan:

Nilai harapan informasi bersifat non-negatif:

$$\forall e, E_j \quad VPI_e(E_j) \geq 0 .$$

Teorema ini mengikuti langsung dari definisi VPI, dan kita tinggalkan pembuktiannya sebagai latihan. Tentu saja, ini adalah teorema tentang nilai yang diharapkan, bukan nilai aktual. Informasi tambahan dapat dengan mudah mengarah pada rencana yang ternyata lebih buruk daripada rencana awal jika informasinya ternyata menyesatkan. Misalnya, tes medis yang memberikan hasil positif palsu dapat menyebabkan pembedahan yang tidak perlu; tetapi itu tidak berarti bahwa tes tersebut tidak boleh dilakukan.

Penting untuk diingat bahwa VPI bergantung pada status informasi terkini, sehingga diberi subskrip. Nilainya dapat berubah seiring bertambahnya informasi yang diperoleh. Untuk setiap bukti E_j , nilai perolehannya dapat turun (misalnya, jika variabel lain sangat membatasi posterior untuk E_j) atau naik (misalnya, jika variabel lain memberikan petunjuk tentang E_j mana yang sedang dibangun, sehingga memungkinkan rencana baru yang lebih baik untuk disusun). Dengan demikian, VPI tidak bersifat aditif. Artinya,

$$VPI_e(E_j, E_k) \neq VPI_e(E_j) + VPI_e(E_k) \quad (\text{umumnya})$$

Namun, VPI bersifat independen terhadap pesanan. Artinya,

$$VPI_e(E_j, E_k) = VPI_e(E_j) + VPI_{e,ej}(E_k) = VPI_e(E_k) + VPI_{e,ek}(E_j) .$$

Kemandirian urutan membedakan tindakan penginderaan dari tindakan biasa dan menyederhanakan masalah penghitungan nilai dari serangkaian tindakan penginderaan.

Implementasi Agen Pengumpul Informasi

Agen yang bijaksana harus mengajukan pertanyaan dalam urutan yang wajar, menghindari pertanyaan yang tidak relevan, mempertimbangkan pentingnya setiap informasi dalam kaitannya dengan biayanya, dan berhenti bertanya ketika diperlukan. Semua kemampuan ini dapat dicapai dengan menggunakan nilai informasi sebagai panduan.

Gambar 7.9 menunjukkan desain keseluruhan agen yang dapat mengumpulkan informasi secara cerdas sebelum bertindak. Untuk saat ini, kami berasumsi bahwa dengan setiap variabel bukti yang dapat diamati, E_j , terdapat biaya terkait, Biaya (E_j), yang mencerminkan biaya untuk memperoleh bukti melalui pengujian, konsultan, pertanyaan, atau apa pun. Agen meminta apa yang tampaknya merupakan observasi paling efisien dalam hal perolehan utilitas per unit biaya. Kami berasumsi bahwa hasil dari tindakan Permintaan (E_j) adalah bahwa persepsi berikutnya memberikan nilai E_j . Jika tidak ada observasi yang sepadan dengan biayanya, agen memilih tindakan "nyata".

Algoritme agen yang telah kami jelaskan menerapkan suatu bentuk pengumpulan informasi yang disebut miopik. Hal ini karena algoritma tersebut menggunakan rumus VPI secara picik, menghitung nilai informasi seolah-olah hanya satu variabel bukti yang akan diperoleh. Kontrol miopik didasarkan pada ide heuristik yang sama dengan pencarian greedy dan seringkali berhasil dalam praktik.

```

function INFORMATION-GATHERING- AGENT(percept) returns an action
  persistent:  $D$ , a decision network

  integrate percept into  $D$ 
   $j \leftarrow$  the value that maximizes  $VPI(E_j) / Cost(E_j)$ 
  if  $VPI(E_j) > Cost(E_j)$ 
    return  $REQUEST(E_j)$ 
  else return the best action from  $D$ 

```

Gambar 7.9 Desain agen pengumpul informasi sederhana. Agen bekerja dengan memilih observasi dengan nilai informasi tertinggi secara berulang, hingga biaya observasi berikutnya lebih besar daripada manfaat yang diharapkan.

Namun, jika tidak ada satu pun variabel bukti yang akan sangat membantu, agen yang rabun mungkin akan terburu-buru mengambil tindakan, padahal akan lebih baik jika meminta dua variabel atau lebih terlebih dahulu, baru kemudian mengambil tindakan. Pendekatan yang lebih baik dalam situasi ini adalah menyusun rencana bersyarat yang meminta nilai variabel dan mengambil langkah selanjutnya yang berbeda, tergantung pada jawabannya.

Pertimbangan terakhir adalah dampak serangkaian pertanyaan terhadap responden manusia. Orang mungkin merespons serangkaian pertanyaan dengan lebih baik jika pertanyaan tersebut "masuk akal", sehingga beberapa sistem pakar dibangun untuk mempertimbangkan hal ini, mengajukan pertanyaan dalam urutan yang memaksimalkan utilitas total sistem dan manusia, alih-alih urutan yang memaksimalkan nilai informasi.

7.7 SISTEM PAKAR TEORI KEPUTUSAN

Bidang analisis keputusan, yang berkembang pada tahun 1950an dan 1960an, mempelajari penerapan teori keputusan pada permasalahan keputusan aktual. Analisis

keputusan digunakan untuk membantu pengambilan keputusan rasional dalam domain penting yang berisiko tinggi, seperti bisnis, pemerintahan, hukum, strategi militer, diagnosis medis dan kesehatan masyarakat, desain teknik, dan manajemen sumber daya. Proses ini melibatkan studi cermat tentang kemungkinan tindakan dan hasil, serta preferensi yang diberikan pada setiap hasil.

Dalam analisis keputusan, biasanya dibahas dua peran: pembuat keputusan menyatakan preferensi di antara hasil, dan analis keputusan menghitung kemungkinan tindakan dan hasil serta meminta preferensi dari pembuat keputusan untuk menentukan tindakan terbaik. Hingga awal 1980an, tujuan utama analisis keputusan adalah untuk membantu manusia membuat keputusan yang benar-benar mencerminkan preferensi mereka sendiri. Seiring semakin banyaknya proses pengambilan keputusan yang terotomatisasi, analisis keputusan semakin banyak digunakan untuk memastikan bahwa proses otomatis tersebut berfungsi sesuai keinginan.

Penelitian sistem pakar awal berfokus pada menjawab pertanyaan, alih-alih membuat keputusan. Sistem-sistem yang merekomendasikan tindakan alih-alih memberikan opini tentang fakta umumnya melakukannya dengan menggunakan aturan kondisi-tindakan, alih-alih dengan representasi eksplisit dari hasil dan preferensi. Munculnya jaringan Bayesian di akhir 1980an memungkinkan pembangunan sistem skala besar yang menghasilkan inferensi probabilistik yang kuat dari bukti. Penambahan jaringan keputusan berarti bahwa sistem pakar dapat dikembangkan yang merekomendasikan keputusan optimal, yang mencerminkan preferensi agen serta bukti yang tersedia.

Sistem yang menggabungkan utilitas dapat menghindari salah satu jebakan paling umum yang terkait dengan proses konsultasi: membingungkan antara kemungkinan dan kepentingan. Strategi umum dalam sistem pakar medis awal, misalnya, adalah memberi peringkat kemungkinan diagnosis berdasarkan kemungkinan dan melaporkan yang paling mungkin. Sayangnya, ini bisa menjadi bencana! Bagi sebagian besar pasien dalam praktik umum, dua diagnosis yang paling mungkin biasanya adalah "Tidak ada yang salah dengan Anda" dan "Anda sedang flu berat," tetapi jika diagnosis ketiga yang paling mungkin untuk pasien tertentu adalah kanker paru-paru, itu masalah serius. Tentu saja, rencana pengujian atau perawatan harus bergantung pada probabilitas dan utilitas. Sistem pakar medis saat ini dapat mempertimbangkan nilai informasi untuk merekomendasikan tes, dan kemudian menjelaskan diagnosis banding.

Kami sekarang menjelaskan proses rekayasa pengetahuan untuk sistem pakar teoretis-keputusan. Sebagai contoh, kami mempertimbangkan masalah pemilihan perawatan medis untuk suatu jenis penyakit jantung bawaan pada anak-anak. Sekitar 0,8% anak lahir dengan anomali jantung, yang paling umum adalah koarktasio aorta (penyempitan aorta). Kondisi ini dapat diobati dengan operasi, angioplasti (memperluas aorta dengan balon yang ditempatkan di dalam arteri), atau pengobatan. Masalahnya adalah memutuskan perawatan apa yang akan digunakan dan kapan melakukannya: semakin muda bayi, semakin besar risiko perawatan tertentu, tetapi seseorang tidak boleh menunggu terlalu lama. Sistem pakar teoretis-keputusan untuk masalah ini dapat dibuat oleh tim yang terdiri dari setidaknya satu pakar

domain (dokter jantung anak) dan satu insinyur pengetahuan. Proses ini dapat dipecah menjadi langkah-langkah berikut:

Membuat model kausal. Menentukan kemungkinan gejala, gangguan, perawatan, dan hasil. Kemudian, gambarlah busur di antara keduanya, yang menunjukkan gangguan apa yang menyebabkan gejala apa, dan perawatan apa yang meringankan gangguan apa. Sebagian dari hal ini akan diketahui oleh pakar di bidang tersebut, dan sebagian lagi akan berasal dari literatur. Seringkali model tersebut akan cocok dengan deskripsi grafis informal yang terdapat dalam buku teks kedokteran.

Sederhanakan menjadi model keputusan kualitatif. Karena kita menggunakan model ini untuk membuat keputusan pengobatan dan bukan untuk tujuan lain (seperti menentukan probabilitas gabungan dari kombinasi gejala/gangguan tertentu), kita seringkali dapat menyederhanakannya dengan menghilangkan variabel yang tidak terlibat dalam keputusan pengobatan.

Terkadang variabel harus dipisah atau digabungkan agar sesuai dengan intuisi pakar. Misalnya, model koarktasio aorta asli memiliki variabel Pengobatan dengan nilai operasi, angioplasti, dan pengobatan, serta variabel terpisah untuk Waktu pengobatan. Namun, pakar kesulitan memikirkan hal-hal ini secara terpisah, sehingga keduanya digabungkan, dengan Pengobatan yang mengambil nilai seperti operasi dalam 1 bulan. Ini menghasilkan model pada Gambar 7.10.

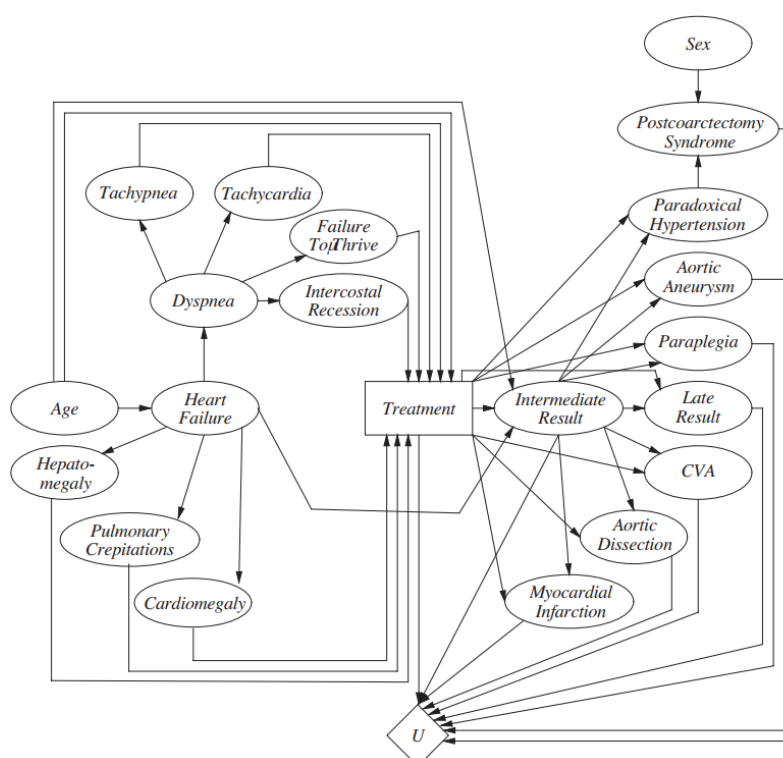
Tetapkan probabilitas. Probabilitas dapat berasal dari basis data pasien, studi literatur, atau penilaian subjektif pakar. Perlu dicatat bahwa sistem diagnostik akan bernalar dari gejala dan observasi lain hingga ke penyakit atau penyebab lain dari masalah tersebut. Oleh karena itu, pada tahun-tahun awal pembangunan sistem ini, para pakar ditanya tentang probabilitas suatu sebab berdasarkan suatu akibat. Umumnya, mereka merasa hal ini sulit dilakukan, dan lebih mampu menilai probabilitas suatu akibat berdasarkan suatu sebab. Oleh karena itu, sistem modern biasanya menilai pengetahuan kausal dan mengodekannya langsung dalam struktur jaringan Bayesian model, menyerahkan penalaran diagnostik kepada algoritma inferensi jaringan Bayesian.

Tetapkan utilitas. Ketika terdapat sejumlah kecil kemungkinan hasil, hasil tersebut dapat dihitung dan dievaluasi secara individual. Kami akan membuat skala dari hasil terbaik hingga terburuk dan memberikan masing-masing nilai numerik, misalnya 0 untuk kematian dan 1 untuk pemulihan total. Kemudian, kami akan menempatkan hasil lainnya pada skala ini. Hal ini dapat dilakukan oleh pakar, tetapi akan lebih baik jika pasien (atau dalam kasus bayi, orang tua pasien) dapat dilibatkan, karena setiap orang memiliki preferensi yang berbeda. Jika terdapat banyak hasil secara eksponensial, kita memerlukan cara untuk menggabungkannya menggunakan fungsi utilitas multiatribut. Misalnya, kita dapat mengatakan bahwa biaya berbagai komplikasi bersifat aditif.

Verifikasi dan sempurnakan modelnya. Untuk mengevaluasi sistem, kita memerlukan serangkaian pasangan (masukan, keluaran) yang benar; yang disebut standar emas untuk dibandingkan. Untuk sistem pakar medis, hal ini biasanya berarti mengumpulkan dokter

terbaik yang tersedia, menyajikan beberapa kasus kepada mereka, dan menanyakan diagnosis serta rencana perawatan yang direkomendasikan.

Kemudian, kita melihat seberapa baik sistem tersebut sesuai dengan rekomendasi mereka. Jika hasilnya buruk, kita mencoba mengisolasi bagian-bagian yang bermasalah dan memperbaikinya. Menjalankan sistem secara "mundur" dapat bermanfaat. Alih-alih menyajikan sistem dengan gejala dan meminta diagnosis, kita dapat menyajikannya dengan diagnosis seperti "gagal jantung", memeriksa probabilitas gejala yang diprediksi seperti takikardia, dan membandingkannya dengan literatur medis.



Gambar 7.10 Diagram pengaruh terhadap koarktasio aorta (milik Peter Lucas).

Lakukan analisis sensitivitas. Langkah penting ini memeriksa apakah keputusan terbaik sensitif terhadap perubahan kecil dalam probabilitas dan utilitas yang ditetapkan dengan memvariasikan parameter tersebut secara sistematis dan menjalankan evaluasi kembali. Jika perubahan kecil menghasilkan keputusan yang berbeda secara signifikan, maka akan bermanfaat untuk mengalokasikan lebih banyak sumber daya untuk mengumpulkan data yang lebih baik. Jika semua variasi menghasilkan keputusan yang sama, maka agen akan lebih yakin bahwa keputusan tersebut tepat.

Analisis sensitivitas sangat penting, karena salah satu kritik utama terhadap pendekatan probabilistik pada sistem pakar adalah terlalu sulitnya menilai probabilitas numerik yang diperlukan. Analisis sensitivitas sering kali mengungkapkan bahwa banyak angka hanya perlu ditentukan secara perkiraan. Misalnya, kita mungkin tidak yakin tentang probabilitas kondisional $P(\text{takikardia} | \text{dispnea})$, tetapi jika keputusan optimal cukup

robust terhadap variasi kecil dalam probabilitas, maka ketidaktahuan kita tidak terlalu menjadi masalah.

7.8 RINGKASAN

Bab ini menunjukkan cara menggabungkan teori utilitas dengan probabilitas untuk memungkinkan agen memilih tindakan yang akan memaksimalkan kinerja yang diharapkan.

- Teori probabilitas menjelaskan apa yang seharusnya diyakini agen berdasarkan bukti, teori utilitas menjelaskan apa yang diinginkan agen, dan teori keputusan menggabungkan keduanya untuk menjelaskan apa yang seharusnya dilakukan agen.
- Kita dapat menggunakan teori keputusan untuk membangun sistem yang membuat keputusan dengan mempertimbangkan semua tindakan yang mungkin dan memilih tindakan yang mengarah pada hasil terbaik yang diharapkan. Sistem semacam itu dikenal sebagai agen rasional.
- Teori utilitas menunjukkan bahwa agen yang preferensinya di antara lotere konsisten dengan serangkaian aksioma sederhana dapat digambarkan memiliki fungsi utilitas; lebih lanjut, agen memilih tindakan seolah-olah memaksimalkan utilitas yang diharapkan.
- Teori utilitas multiatribut membahas utilitas yang bergantung pada beberapa atribut keadaan yang berbeda. Dominasi stokastik merupakan teknik yang sangat berguna untuk membuat keputusan yang tidak ambigu, bahkan tanpa nilai utilitas yang tepat untuk atribut.
- Jaringan keputusan menyediakan formalisme sederhana untuk mengekspresikan dan memecahkan masalah keputusan. Jaringan ini merupakan perluasan alami dari jaringan Bayesian, yang berisi simpul keputusan dan utilitas di samping simpul peluang.
- Terkadang, memecahkan masalah melibatkan pencarian informasi lebih lanjut sebelum membuat keputusan. Nilai informasi didefinisikan sebagai peningkatan utilitas yang diharapkan dibandingkan dengan membuat keputusan tanpa informasi tersebut.
- Sistem pakar yang menggabungkan informasi utilitas memiliki kemampuan tambahan dibandingkan dengan sistem inferensi murni. Selain mampu membuat keputusan, sistem pakar dapat menggunakan nilai informasi untuk memutuskan pertanyaan mana yang akan diajukan, jika ada; sistem pakar dapat merekomendasikan rencana kontingensi; dan sistem pakar dapat menghitung sensitivitas keputusan mereka terhadap perubahan kecil dalam penilaian probabilitas dan utilitas.

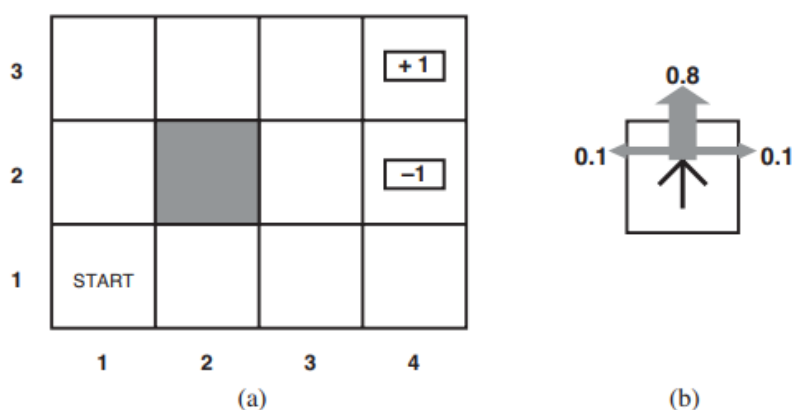
BAB 8

MEMBUAT KEPUTUSAN KOMPLEKS

8.1 MASALAH KEPUTUSAN SEKUENSIAL

Misalkan sebuah agen berada dalam lingkungan 4×3 yang ditunjukkan pada Gambar 8.1(a). Dimulai dari keadaan awal, agen tersebut harus memilih tindakan pada setiap langkah waktu. Interaksi dengan lingkungan berakhir ketika agen mencapai salah satu status tujuan, ditandai $+1$ atau -1 . Sebagaimana halnya masalah pencarian, tindakan yang tersedia bagi agen di setiap status ditentukan oleh $ACTION(s)$, terkadang disingkat menjadi $A(s)$; dalam lingkungan 4×3 , tindakan di setiap status adalah *Up*, *Down*, *Left*, dan *Right*. Untuk saat ini, kami berasumsi bahwa lingkungan sepenuhnya dapat diamati, sehingga agen selalu tahu di mana ia berada.

Jika lingkungannya deterministik, solusinya akan mudah: [*Up*, *Up*, *Right*, *Right*, *Right*]. Sayangnya, lingkungan tidak akan selalu mendukung solusi ini, karena tindakannya tidak dapat diandalkan. Model khusus gerak stokastik yang kami adopsi diilustrasikan pada Gambar 8.1(b). Setiap tindakan mencapai efek yang diinginkan dengan probabilitas 0,8, tetapi di waktu lainnya, tindakan tersebut menggerakkan agen tegak lurus ke arah yang diinginkan. Lebih lanjut, jika agen menabrak dinding, ia tetap berada di kotak yang sama. Misalnya, dari kotak awal (1,1), tindakan Naik menggerakkan agen ke (1,2) dengan probabilitas 0,8, tetapi dengan probabilitas 0,1, ia bergerak ke kanan ke (2,1), dan dengan probabilitas 0,1, ia bergerak ke kiri, menabrak dinding, dan tetap di (1,1). Dalam lingkungan seperti itu, urutan [*Up*, *Up*, *Right*, *Right*, *Right*] naik di sekitar penghalang dan mencapai status tujuan di (4,3) dengan probabilitas $0,8^5 = 0,32768$. Ada juga peluang kecil untuk secara tidak sengaja mencapai tujuan dengan bergerak sebaliknya dengan probabilitas $0,1^4 \times 0,8$, dengan total keseluruhan 0,32776.



Gambar 8.1 (a) Lingkungan 4×3 sederhana yang menyajikan agen dengan masalah keputusan sekuensial. (b) Ilustrasi model transisi lingkungan: hasil "yang diinginkan" terjadi dengan probabilitas 0,8, tetapi dengan probabilitas 0,2 agen bergerak tegak lurus ke arah yang diinginkan. Tabrakan dengan dinding tidak menghasilkan pergerakan. Kedua status

terminal memiliki imbalan +1 dan -1, masing-masing, dan semua status lainnya memiliki imbalan -0,04.

Seperti dalam Bab 3, model transisi (atau hanya "model," kapan pun tidak ada kebingungan yang muncul) menggambarkan hasil dari setiap tindakan di setiap status. Di sini, hasilnya stokastik, jadi kami menulis $P(s' | s, a)$ untuk menunjukkan probabilitas mencapai status s' jika tindakan a dilakukan di status s . Kami akan berasumsi bahwa transisi bersifat Markovian dalam pengertian Bab 15, yaitu, probabilitas mencapai s' dari s hanya bergantung pada s dan bukan pada riwayat status sebelumnya. Untuk saat ini, Anda dapat menganggap $P(s' | s, a)$ sebagai tabel tiga dimensi besar yang memuat probabilitas. Nanti kita akan melihat bahwa model transisi dapat direpresentasikan sebagai jaringan Bayesian yang dinamis.

Untuk melengkapi definisi lingkungan tugas, kita harus menentukan fungsi utilitas untuk agen. Karena masalah keputusan bersifat berurutan, fungsi utilitas akan bergantung pada urutan status—riwayat lingkungan—daripada pada satu status. Nanti di bagian ini, kita menyelidiki bagaimana fungsi utilitas tersebut dapat ditentukan secara umum; untuk saat ini, kita cukup menetapkan bahwa di setiap status s , agen menerima imbalan $R(s)$, yang mungkin positif atau negatif, tetapi harus dibatasi. Untuk contoh khusus kita, imbalannya adalah -0,04 di semua status kecuali status terminal (yang memiliki imbalan +1 dan -1). Utilitas riwayat lingkungan hanyalah (untuk saat ini) jumlah imbalan yang diterima. Misalnya, jika agen mencapai keadaan +1 setelah 10 langkah, utilitas totalnya akan menjadi 0,6. Imbalan negatif sebesar -0,04 memberi agen insentif untuk mencapai (4,3) dengan cepat, sehingga lingkungan kita merupakan generalisasi stokastik. Cara lain untuk menyatakan hal ini adalah bahwa agen tidak menikmati hidup di lingkungan ini dan ingin segera pergi.

Singkatnya: masalah keputusan sekuensial untuk lingkungan stokastik yang sepenuhnya dapat diamati dengan model transisi Markovian dan imbalan aditif disebut proses keputusan Markov, atau MDP, dan terdiri dari serangkaian keadaan (dengan keadaan awal s_0); serangkaian *ACTION* tindakan di setiap keadaan; model transisi $P(s' | s, a)$; dan fungsi imbalan $R(s)$ ¹.

Pertanyaan selanjutnya adalah, seperti apa solusi untuk masalah ini? Kita telah melihat bahwa urutan tindakan yang tetap tidak akan menyelesaikan masalah, karena agen mungkin berakhir di keadaan selain tujuan. Oleh karena itu, sebuah solusi harus menentukan apa yang harus dilakukan agen untuk setiap keadaan yang mungkin dicapainya. Solusi semacam ini disebut kebijakan. Secara tradisional, kebijakan dilambangkan dengan π , dan $\pi(s)$ adalah tindakan yang direkomendasikan oleh kebijakan π untuk keadaan s . Jika agen memiliki kebijakan yang lengkap, apa pun hasil dari tindakan tersebut, agen akan selalu tahu apa yang harus dilakukan selanjutnya.

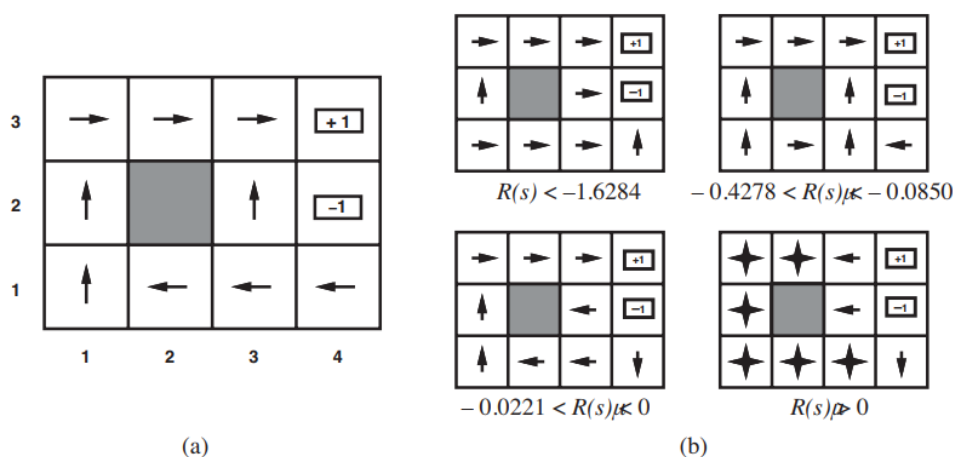
Setiap kali suatu kebijakan dieksekusi mulai dari keadaan awal, sifat stokastik lingkungan dapat menghasilkan riwayat lingkungan yang berbeda. Oleh karena itu, kualitas suatu kebijakan diukur berdasarkan utilitas yang diharapkan dari kemungkinan riwayat lingkungan yang dihasilkan oleh kebijakan tersebut. Kebijakan optimal adalah kebijakan yang menghasilkan utilitas yang diharapkan tertinggi. Kami menggunakan π^* untuk menyatakan

suatu kebijakan optimal. Dengan π^* , agen memutuskan apa yang harus dilakukan dengan merujuk pada persepsinya saat ini, yang memberi tahu keadaan saat ini s , dan kemudian mengeksekusi tindakan $\pi^*(s)$. Suatu kebijakan merepresentasikan fungsi agen secara eksplisit dan oleh karena itu merupakan deskripsi agen refleks sederhana, yang dihitung dari informasi yang digunakan untuk agen berbasis utilitas.

Kebijakan optimal untuk dunia pada Gambar 8.1 ditunjukkan pada Gambar 8.2(a). Perhatikan bahwa, karena biaya untuk mengambil suatu langkah cukup kecil dibandingkan dengan penalti karena berakhir di (4,2) secara tidak sengaja, kebijakan optimal untuk keadaan (3,1) bersifat konservatif. Kebijakan tersebut merekomendasikan untuk mengambil jalan memutar, daripada mengambil jalan pintas dan dengan demikian mengambil risiko memasuki (4,2).

Keseimbangan risiko dan imbalan berubah tergantung pada nilai $R(s)$ untuk keadaan nonterminal. Gambar 8.2(b) menunjukkan kebijakan optimal untuk empat rentang $R(s)$ yang berbeda. Ketika $R(s) \leq -1,6284$, hidup terasa begitu menyakitkan sehingga agen langsung menuju jalan keluar terdekat, meskipun jalan keluar tersebut bernilai -1 . Ketika $-0,4278 \leq R(s) \leq -0,0850$, hidup terasa sangat tidak menyenangkan; agen mengambil rute terpendek menuju keadaan $+1$ dan bersedia mengambil risiko jatuh ke keadaan -1 secara tidak sengaja.

Secara khusus, agen mengambil jalan pintas dari (3,1). Ketika hidup hanya sedikit suram ($-0,0221 < R(s) < 0$), kebijakan optimal tidak mengambil risiko sama sekali. Dalam (4,1) dan (3,2), agen langsung menjauh dari keadaan -1 agar tidak jatuh secara tidak sengaja, meskipun ini berarti membenturkan kepalanya ke dinding beberapa kali. Terakhir, jika $R(s) > 0$, maka hidup terasa menyenangkan dan agen menghindari kedua jalan keluar tersebut. Selama tindakan dalam (4,1), (3,2), dan (3,3) seperti yang ditunjukkan, setiap kebijakan optimal, dan agen memperoleh total imbalan tak terhingga karena tidak pernah memasuki keadaan terminal. Anehnya, ternyata terdapat enam kebijakan optimal lainnya untuk berbagai rentang $R(s)$.



Gambar 8.2 (a) Kebijakan optimal untuk lingkungan stokastik dengan $R(s) = -0,04$ pada keadaan nonterminal. (b) Kebijakan optimal untuk empat rentang $R(s)$ yang berbeda.

Penyeimbangan risiko dan imbalan yang cermat merupakan karakteristik MDP yang tidak muncul dalam masalah pencarian deterministik; lebih lanjut, hal ini merupakan karakteristik dari banyak masalah keputusan di dunia nyata. Karena alasan ini, MDP telah dipelajari di beberapa bidang, termasuk AI, riset operasi, ekonomi, dan teori kendali. Puluhan algoritma telah diusulkan untuk menghitung kebijakan optimal. Pada bagian selanjutnya, kami menjelaskan dua keluarga algoritma yang paling penting. Namun, pertama-tama kita harus menyelesaikan penyelidikan kita terhadap utilitas dan kebijakan untuk masalah keputusan berurutan.

Utilitas dari Waktu ke Waktu

Dalam contoh MDP pada Gambar 8.1, kinerja agen diukur dengan jumlah imbalan untuk status yang dikunjungi. Pilihan ukuran kinerja ini tidak sembarangan, tetapi bukan satu-satunya kemungkinan untuk fungsi utilitas pada riwayat lingkungan, yang kami tulis sebagai $U_h([s_0, s_1, \dots, s_n])$. Analisis kami mengacu pada teori utilitas multiatribut dan agak teknis; pembaca yang tidak sabar mungkin ingin langsung ke bagian berikutnya.

Pertanyaan pertama yang harus dijawab adalah apakah ada horizon terbatas atau horizon tak terbatas untuk pengambilan keputusan. Horizon terbatas berarti ada waktu tetap N yang setelahnya tidak ada yang penting—permainan berakhir, begitulah istilahnya. Jadi, $U_h([s_0, s_1, \dots, s_{N+k}]) = U_h([s_0, s_1, \dots, s_N])$ untuk semua $k > 0$. Misalnya, anggaplah seorang agen memulai pada (3,1) di dunia 4×3 pada Gambar 8.1, dan anggaplah bahwa $N = 3$. Kemudian, untuk memiliki peluang mencapai status +1, agen harus langsung menuju ke sana, dan tindakan optimalnya adalah Naik. Di sisi lain, jika $N = 100$, maka ada banyak waktu untuk mengambil rute aman dengan ke *Left*.

Jadi, dengan horizon terbatas, tindakan optimal dalam status tertentu dapat berubah seiring waktu. Kami katakan bahwa kebijakan optimal untuk horizon terbatas adalah nonstasioner. Di sisi lain, tanpa batas waktu yang tetap, tidak ada alasan untuk berperilaku berbeda dalam keadaan yang sama pada waktu yang berbeda. Oleh karena itu, tindakan optimal hanya bergantung pada keadaan saat ini, dan kebijakan optimal bersifat stasioner.

Oleh karena itu, kebijakan untuk kasus horizon tak terbatas lebih sederhana daripada kebijakan untuk kasus horizon terbatas, dan kami terutama membahas kasus horizon tak terbatas dalam bab ini. (Kita akan melihat nanti bahwa untuk lingkungan yang dapat diamati sebagian, kasus horizon tak terbatas tidak sesederhana itu.) Perhatikan bahwa "horison tak terbatas" tidak selalu berarti bahwa semua urutan keadaan tidak terbatas; itu hanya berarti bahwa tidak ada batas waktu yang tetap. Secara khusus, dapat ada urutan keadaan terbatas dalam MDP horizon tak terbatas yang berisi keadaan terminal.

Pertanyaan berikutnya yang harus kita putuskan adalah bagaimana menghitung utilitas dari urutan keadaan. Dalam terminologi teori utilitas multiatribut, setiap keadaan s_i dapat dilihat sebagai atribut dari urutan keadaan $[s_0, s_1, s_2 \dots]$. Untuk memperoleh ekspresi sederhana dalam hal atribut, kita perlu membuat semacam asumsi independensi-preferensi. Asumsi yang paling alami adalah bahwa preferensi agen antara urutan keadaan bersifat stasioner. Stasioneritas untuk preferensi berarti sebagai berikut: jika dua urutan keadaan $[s_0, s_1, s_2 \dots]$ dan $[s_0^1, s_1^1, s_2^1, \dots]$ dimulai dengan keadaan yang sama (yaitu, $s_0 = s_0^1$), maka kedua

urutan tersebut harus diurutkan preferensinya dengan cara yang sama seperti urutan $[s_1, s_2, \dots]$ dan $[s_1^I, s_2^I, \dots]$.

Dalam bahasa Inggris, ini berarti jika Anda lebih menyukai satu masa depan daripada masa depan lain yang dimulai besok, maka Anda tetap harus lebih menyukai masa depan tersebut jika dimulai hari ini. Stasioneritas adalah asumsi yang tampak cukup tidak berbahaya dengan konsekuensi yang sangat kuat: ternyata dalam stasioneritas hanya ada dua cara yang koheren untuk menetapkan utilitas pada deret:

1. Imbalan aditif: Utilitas deret keadaan adalah

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

Dunia 4×3 pada Gambar 8.1 menggunakan imbalan aditif. Perhatikan bahwa aditivitas digunakan secara implisit dalam penggunaan fungsi biaya jalur dalam algoritma pencarian heuristik.

2. Imbalan yang didiskon: Kegunaan urutan keadaan adalah

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

dengan faktor diskonto γ adalah angka antara 0 dan 1. Faktor diskonto menggambarkan preferensi agen terhadap imbalan saat ini dibandingkan imbalan di masa mendatang. Ketika γ mendekati 0, imbalan di masa mendatang dianggap tidak signifikan. Ketika γ bernilai 1, imbalan yang didiskontokan setara persis dengan imbalan aditif, sehingga imbalan aditif merupakan kasus khusus dari imbalan yang didiskontokan. Pendiskontoan tampaknya menjadi model yang baik untuk preferensi hewan dan manusia dari waktu ke waktu. Faktor diskonto γ setara dengan suku bunga $(1/\gamma) - 1$. Untuk alasan yang akan segera dijelaskan, kami mengasumsikan imbalan yang didiskontokan di sisa bab ini, meskipun terkadang kami mengizinkan $\gamma = 1$.

Di balik pilihan cakrawala tak terbatas kami terdapat sebuah masalah: jika lingkungan tidak mengandung keadaan terminal, atau jika agen tidak pernah mencapainya, maka semua riwayat lingkungan akan sangat panjang, dan utilitas dengan imbalan aditif yang tidak didiskontokan umumnya akan tak terbatas. Meskipun kita dapat sepakat bahwa $+\infty$ lebih baik daripada $-\infty$, membandingkan dua deret keadaan dengan utilitas $+\infty$ lebih sulit. Ada tiga solusi, dua di antaranya telah kita lihat:

1. Dengan imbalan yang didiskontokan, utilitas deret tak terhingga adalah terbatas. Faktanya, jika $\gamma < 1$ dan imbalan dibatasi oleh $\pm R_{max}$, kita memiliki

$$U_h([s_0, s_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1 - \gamma}, \quad (8.1)$$

menggunakan rumus standar untuk jumlah deret geometri tak terhingga.

2. Jika lingkungan berisi status terminal dan jika agen dijamin akan mencapainya pada akhirnya, maka kita tidak perlu membandingkan deret tak terhingga. Kebijakan yang dijamin mencapai status terminal disebut kebijakan yang tepat. Dengan kebijakan yang tepat, kita dapat menggunakan $\gamma = 1$ (yaitu, imbalan aditif). Tiga kebijakan pertama yang ditunjukkan pada Gambar 8.2(b) adalah tepat, tetapi yang keempat tidak tepat. Kebijakan ini memperoleh imbalan total tak terhingga dengan menjauhi status terminal saat imbalan untuk status nonterminal positif. Adanya kebijakan yang tidak tepat dapat menyebabkan algoritme standar untuk memecahkan MDP gagal dengan imbalan aditif, dan dengan demikian memberikan alasan yang baik untuk menggunakan imbalan yang didiskon.
3. Deret tak terhingga dapat dibandingkan dalam hal imbalan rata-rata yang diperoleh per langkah waktu. Misalkan kuadrat (1,1) di dunia 4×3 memiliki imbalan 0,1 sedangkan status nonterminal lainnya memiliki imbalan 0,01. Maka kebijakan yang berusaha sebaik mungkin untuk tetap berada di (1,1) akan memiliki imbalan rata-rata yang lebih tinggi daripada kebijakan yang tetap berada di tempat lain. Imbalan rata-rata merupakan kriteria yang berguna untuk beberapa masalah, tetapi analisis algoritma imbalan rata-rata berada di luar cakupan buku ini.

Singkatnya, imbalan yang didiskontokan menghadirkan kesulitan paling sedikit dalam mengevaluasi urutan status.

Kebijakan Optimal dan Utilitas Status

Setelah memutuskan bahwa utilitas urutan status tertentu adalah jumlah imbalan yang didiskontokan yang diperoleh selama urutan tersebut, kita dapat membandingkan kebijakan dengan membandingkan utilitas yang diharapkan yang diperoleh saat menjalankannya. Kita berasumsi agen berada dalam beberapa status awal s dan mendefinisikan S_t (variabel acak) sebagai status yang dicapai agen pada waktu t saat menjalankan kebijakan tertentu π . (Jelas, $S_0 = s$, status agen saat ini.) Distribusi probabilitas atas urutan status $S_1, S_2, \dots$, ditentukan oleh status awal s , kebijakan π , dan model transisi untuk lingkungan.

Utilitas yang diharapkan diperoleh dengan menjalankan π mulai dari s diberikan oleh

$$U^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right], \quad (8.2)$$

di mana ekspektasinya berkaitan dengan distribusi probabilitas atas deret keadaan yang ditentukan oleh s dan π . Sekarang, dari semua kebijakan yang dapat dipilih agen untuk dijalankan mulai dari s , satu (atau lebih) akan memiliki utilitas ekspektasi yang lebih tinggi daripada yang lainnya. Kita akan menggunakan π_g^* untuk menunjukkan salah satu kebijakan ini:

$$\pi_g^* = \underset{\pi}{argmax} U^\pi(s) \quad (8.3)$$

Ingat bahwa π_g^* adalah sebuah kebijakan, jadi ia merekomendasikan tindakan untuk setiap keadaan; hubungannya dengan s khususnya adalah bahwa ia adalah kebijakan optimal ketika s adalah keadaan awal. Konsekuensi luar biasa dari penggunaan utilitas yang didiskontokan dengan cakrawala tak terbatas adalah bahwa kebijakan optimal tidak bergantung pada keadaan awal. (Tentu saja, urutan tindakan tidak akan independen; ingat bahwa kebijakan adalah fungsi yang menentukan tindakan untuk setiap keadaan.) Fakta ini tampak jelas secara intuitif: jika kebijakan π_a^* optimal dimulai di a dan kebijakan π_b^* optimal dimulai di b , maka, ketika mereka mencapai keadaan ketiga c , tidak ada alasan yang baik bagi mereka untuk tidak setuju satu sama lain, atau dengan π_c^* , tentang apa yang harus dilakukan selanjutnya. Jadi kita dapat dengan mudah menulis π^* untuk kebijakan optimal.

Dengan definisi ini, utilitas sebenarnya dari suatu keadaan hanyalah $U^{\pi^*}(s)$ —yaitu, jumlah yang diharapkan dari imbalan yang didiskontokan jika agen menjalankan kebijakan optimal. Kita menuliskannya sebagai $U(s)$, sesuai dengan notasi yang digunakan dalam Bab 16 untuk utilitas suatu hasil. Perhatikan bahwa $U(s)$ dan $R(s)$ adalah kuantitas yang sangat berbeda; $R(s)$ adalah imbalan "jangka pendek" karena berada di s , sedangkan $U(s)$ adalah imbalan total "jangka panjang" dari s dan seterusnya. Gambar 8.3 menunjukkan utilitas untuk dunia 4×3 . Perhatikan bahwa utilitas lebih tinggi untuk keadaan yang lebih dekat dengan pintu keluar $+1$, karena lebih sedikit langkah yang diperlukan untuk mencapai pintu keluar.

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	- 1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Gambar 8.3 Utilitas keadaan di dunia 4×3 , dihitung dengan $\gamma = 1$ dan $R(s) = -0,04$ untuk keadaan nonterminal.

Fungsi utilitas $U(s)$ memungkinkan agen untuk memilih tindakan dengan menggunakan prinsip utilitas yang diharapkan maksimum yaitu, memilih tindakan yang memaksimalkan utilitas yang diharapkan dari keadaan berikutnya:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' | s, a) U(s'). \quad (8.4)$$

Dua bagian berikutnya menjelaskan algoritma untuk menemukan kebijakan optimal.

8.2 ITERASI NILAI

Pada bagian ini, kami menyajikan sebuah algoritma, yang disebut iterasi nilai, untuk menghitung kebijakan yang optimal. Ide dasarnya adalah menghitung utilitas setiap keadaan dan kemudian menggunakan utilitas keadaan untuk memilih tindakan yang optimal di setiap keadaan.

Persamaan Bellman untuk Utilitas

Bagian ini mendefinisikan utilitas berada dalam suatu keadaan sebagai jumlah yang diharapkan dari imbalan yang didiskontokan sejak saat itu dan seterusnya. Dari sini, dapat disimpulkan bahwa ada hubungan langsung antara utilitas suatu keadaan dan utilitas tetangganya: utilitas suatu keadaan adalah imbalan langsung untuk keadaan itu ditambah utilitas yang didiskontokan yang diharapkan dari keadaan berikutnya, dengan asumsi bahwa agen memilih tindakan yang optimal. Artinya, utilitas suatu keadaan diberikan oleh

$$U(s) = R(s) + \gamma \max_{a \in A(a)} \sum_{s'} P(s' | s, a) U(s') . \quad (8.5)$$

Persamaan ini disebut persamaan Bellman, diambil dari Richard Bellman. Utilitas keadaan—yang didefinisikan oleh Persamaan (8.2) sebagai utilitas yang diharapkan dari rangkaian keadaan selanjutnya—merupakan solusi dari himpunan persamaan Bellman. Faktanya, persamaan-persamaan ini merupakan solusi unik. Mari kita lihat salah satu persamaan Bellman untuk dunia 4×3 . Persamaan untuk keadaan (1,1) adalah

$$U(1,1) = -0.04 + \gamma \max \begin{cases} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), & (Up) \\ 0.9U(1,1) + 0.1U(1,2), & (Left) \\ 0.9U(1,1) + 0.1U(2,1), & (Down) \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \end{cases} \quad (Right)$$

Ketika kita memasukkan angka-angka dari Gambar 8.3, kita menemukan bahwa Naik adalah tindakan terbaik.

Algoritma Iterasi Nilai

Persamaan Bellman adalah dasar dari algoritma iterasi nilai untuk memecahkan MDP. Jika ada n kemungkinan keadaan, maka ada n persamaan Bellman, satu untuk setiap keadaan. Persamaan n berisi n variabel yang tidak diketahui—utilitas keadaan. Jadi kita ingin memecahkan persamaan simultan ini untuk menemukan utilitas. Ada satu masalah: persamaan tersebut nonlinier, karena operator "maks" bukanlah operator linier. Sementara sistem persamaan linier dapat dipecahkan dengan cepat menggunakan teknik aljabar linier, sistem persamaan nonlinier lebih bermasalah.

Satu hal yang dapat dicoba adalah pendekatan iteratif. Kita mulai dengan nilai awal yang sembarang untuk utilitas, menghitung sisi kanan persamaan, dan memasukkannya ke sisi kiri—dengan demikian memperbarui utilitas setiap keadaan dari utilitas tetangganya. Kita ulangi ini hingga kita mencapai keseimbangan. Misalkan $U_i(s)$ adalah nilai utilitas untuk

keadaan s pada iterasi ke- i . Langkah iterasi, yang disebut pembaruan Bellman, terlihat seperti ini:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U_i(s'), \quad (8.6)$$

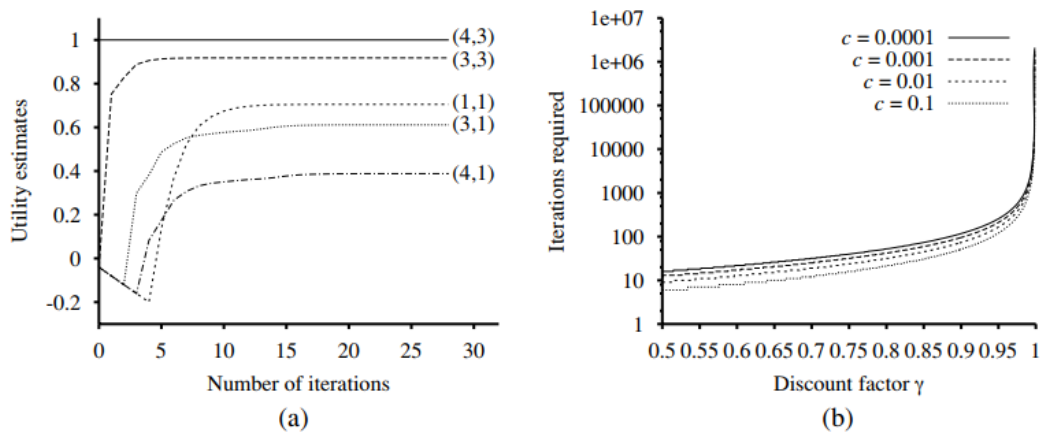
```

function VALUE-ITERATION(mdp,  $\epsilon$ ) returns a utility function
  inputs: mdp, an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,
           rewards  $R(s)$ , discount  $\gamma$ 
            $\epsilon$ , the maximum error allowed in the utility of any state
  local variables:  $U, U'$ , vectors of utilities for states in  $S$ , initially zero
                     $\delta$ , the maximum change in the utility of any state in an iteration

  repeat
     $U \leftarrow U'; \delta \leftarrow 0$ 
    for each state  $s$  in  $S$  do
       $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$ 
      if  $|U[s] - U'[s]| > \delta$  then  $\delta \leftarrow |U[s] - U'[s]|$ 
  until  $\delta < \epsilon(1-\gamma)/\gamma$ 

```

Gambar 8.4 Algoritma iterasi nilai untuk menghitung utilitas keadaan. Kondisi terminasi diambil dari Persamaan (8.8).



Gambar 8.5 (a) Grafik yang menunjukkan evolusi utilitas dari keadaan yang dipilih menggunakan iterasi nilai. (b) Jumlah iterasi nilai k yang diperlukan untuk menjamin kesalahan paling banyak $\epsilon = c \cdot R_{\max}$, untuk nilai c yang berbeda, sebagai fungsi dari faktor diskonto γ .

di mana pembaruan diasumsikan diterapkan secara simultan ke semua keadaan pada setiap iterasi. Jika kita menerapkan pembaruan Bellman tanpa batas frekuensi, kita dijamin mencapai ekuilibrium, yang dalam hal ini nilai utilitas akhir harus menjadi solusi untuk

persamaan Bellman. Bahkan, nilai-nilai tersebut juga merupakan solusi unik, dan kebijakan yang sesuai (diperoleh menggunakan Persamaan (8.4)) adalah optimal. Algoritmanya, yang disebut ITERASI NILAI, ditunjukkan pada Gambar 8.4.

Kita dapat menerapkan iterasi nilai ke dunia 4×3 pada Gambar 8.1(a). Dimulai dengan nilai awal nol, utilitas berevolusi seperti yang ditunjukkan pada Gambar 8.5(a). Perhatikan bagaimana keadaan pada jarak yang berbeda dari (4,3) mengakumulasi imbalan negatif hingga ditemukan jalur ke (4,3), di mana utilitas mulai meningkat. Kita dapat menganggap algoritma iterasi nilai sebagai penyebaran informasi melalui ruang keadaan melalui pembaruan lokal.

Konvergensi Iterasi Nilai

Kita telah membahas bahwa iterasi nilai pada akhirnya akan konvergen menjadi satu set solusi unik dari persamaan Bellman. Di bagian ini, kami akan menjelaskan mengapa hal ini terjadi. Kami memperkenalkan beberapa ide matematika yang berguna, dan kami memperoleh beberapa metode untuk menilai kesalahan dalam fungsi utilitas yang dikembalikan ketika algoritma dihentikan lebih awal; hal ini berguna karena berarti kita tidak perlu menjalankannya selamanya. Bagian ini cukup teknis.

Konsep dasar yang digunakan untuk menunjukkan bahwa iterasi nilai konvergen adalah gagasan kontraksi. Secara kasar, kontraksi adalah fungsi dari satu argumen yang, ketika diterapkan pada dua masukan yang berbeda secara bergantian, menghasilkan dua nilai keluaran yang "lebih dekat satu sama lain," setidaknya dengan beberapa faktor konstan, daripada masukan aslinya. Misalnya, fungsi "bagi dua" adalah kontraksi, karena, setelah kita membagi dua angka dengan dua, selisihnya dibagi dua. Perhatikan bahwa fungsi "bagi dua" memiliki titik tetap, yaitu nol, yang tidak berubah meskipun fungsi tersebut diterapkan. Dari contoh ini, kita dapat melihat dua sifat penting kontraksi:

- Kontraksi hanya memiliki satu titik tetap; jika ada dua titik tetap, keduanya tidak akan saling mendekat ketika fungsi diterapkan, sehingga bukan kontraksi.
- Ketika fungsi diterapkan pada argumen apa pun, nilainya harus mendekati titik tetap (karena titik tetap tidak bergerak), sehingga penerapan kontraksi yang berulang selalu mencapai titik tetap dalam limit.

Sekarang, misalkan kita memandang pembaruan Bellman (Persamaan (8.6)) sebagai operator B yang diterapkan secara simultan untuk memperbarui utilitas setiap keadaan. Misalkan U_i menyatakan vektor utilitas untuk semua keadaan pada iterasi ke- i . Maka persamaan pembaruan Bellman dapat ditulis sebagai

$$U_{i+1} \leftarrow BU_i$$

Selanjutnya, kita membutuhkan cara untuk mengukur jarak antar vektor utilitas. Kita akan menggunakan norma maksimum, yang mengukur "panjang" suatu vektor berdasarkan nilai absolut komponen terbesarnya:

$$\|U\| = \max_s |U(s)|.$$

Dengan definisi ini, “jarak” antara dua vektor, $\|U - U'\|$, adalah selisih maksimum antara dua elemen yang bersesuaian. Hasil utama dari bagian ini adalah sebagai berikut: Misalkan U_i dan U'_1 adalah dua vektor utilitas. Maka kita memiliki

$$\|BU_i - BU'_1\| \leq \gamma \|U_i - U'_1\|. \quad (8.7)$$

Artinya, pembaruan Bellman adalah kontraksi dengan faktor γ pada ruang vektor utilitas. Oleh karena itu, dari sifat-sifat kontraksi secara umum, dapat disimpulkan bahwa iterasi nilai selalu konvergen ke solusi unik persamaan Bellman setiap kali $\gamma < 1$.

Kita juga dapat menggunakan sifat kontraksi untuk menganalisis laju konvergensi menuju suatu solusi. Secara khusus, kita dapat mengganti U'_1 pada Persamaan (8.7) dengan utilitas sebenarnya U , yang mana $BU = U$. Kemudian kita memperoleh pertidaksamaan

$$\|BU_i - U\| \leq \gamma \|U_i - U\|.$$

Jadi, jika kita melihat $\|U_i - U\|$ sebagai kesalahan dalam estimasi U_i , kita melihat bahwa kesalahan berkurang dengan faktor setidaknya γ pada setiap iterasi. Ini berarti bahwa iterasi nilai konvergen secara eksponensial cepat. Kita dapat menghitung jumlah iterasi yang diperlukan untuk mencapai batas kesalahan yang ditentukan ϵ sebagai berikut: Pertama, ingat kembali dari Persamaan (8.1) bahwa utilitas semua keadaan dibatasi oleh $\pm R_{\max}/(1 - \gamma)$. Ini berarti bahwa kesalahan awal maksimum $\|U_0 - U\| \leq 2R_{\max}/(1 - \gamma)$. Misalkan kita menjalankan iterasi N untuk mencapai kesalahan paling banyak ϵ . Kemudian, karena kesalahan berkurang setidaknya γ setiap kali, kita memerlukan $\gamma^N \cdot 2R_{\max}/(1 - \gamma) \leq \epsilon$. Mengambil log, kita menemukan

$$N = \lceil \log(2R_{\max}/\epsilon (1 - \gamma)) / \log(1/\gamma) \rceil$$

iterasi sudah cukup. Gambar 8.5(b) menunjukkan bagaimana N bervariasi dengan γ , untuk nilai rasio $\epsilon/R_{\max}$ yang berbeda. Berita baiknya adalah, karena konvergensi yang sangat cepat, N tidak terlalu bergantung pada rasio $\epsilon/R_{\max}$. Berita buruknya adalah N tumbuh dengan cepat saat γ mendekati 1. Kita bisa mendapatkan konvergensi cepat jika kita membuat γ kecil, tetapi ini secara efektif memberi agen cakrawala yang pendek dan dapat melewatkan efek jangka panjang dari tindakan agen.

Batasan kesalahan dalam paragraf sebelumnya memberikan gambaran tentang faktor-faktor yang memengaruhi waktu berjalan algoritma, tetapi terkadang terlalu konservatif sebagai metode untuk memutuskan kapan harus menghentikan iterasi. Untuk tujuan terakhir, kita dapat menggunakan batasan yang menghubungkan kesalahan dengan ukuran pembaruan Bellman pada setiap iterasi yang diberikan. Dari sifat kontraksi (Persamaan (8.7)), dapat

ditunjukkan bahwa jika pembaruannya kecil (yaitu, utilitas suatu negara tidak banyak berubah), maka kesalahannya, dibandingkan dengan fungsi utilitas sebenarnya, juga kecil. Lebih tepatnya,

$$\text{ jika } \|U_{i+1} - U_i\| < \epsilon (1 - \gamma)/\gamma \text{ kemudian } \|U_{i+1} - U\| < \epsilon. \quad (8.8)$$

Ini adalah kondisi terminasi yang digunakan dalam algoritma VALUE-ITERATION pada Gambar 8.4.

Sejauh ini, kita telah menganalisis kesalahan dalam fungsi utilitas yang dikembalikan oleh algoritma iterasi nilai. Namun, yang benar-benar diperhatikan oleh agen adalah seberapa baik kinerjanya jika ia membuat keputusan berdasarkan fungsi utilitas ini. Misalkan setelah i iterasi iterasi nilai, agen memiliki estimasi U_i dari utilitas U yang sebenarnya dan memperoleh kebijakan MEU π_i berdasarkan one-step look-ahead menggunakan U_i (seperti pada Persamaan (8.4)). Akankah perilaku yang dihasilkan hampir sama baiknya dengan perilaku optimal? Ini adalah pertanyaan krusial bagi setiap agen nyata, dan ternyata jawabannya adalah ya. $U^{\pi_i}(s)$ adalah utilitas yang diperoleh jika π_i dieksekusi mulai dari s , dan kerugian kebijakan $\|U^{\pi_i} - U\|$ adalah kerugian terbesar yang dapat dialami agen dengan mengeksekusi π_i alih-alih kebijakan optimal π^* . Kerugian kebijakan π_i dihubungkan dengan kesalahan dalam U_i melalui pertidaksamaan berikut:

$$\text{ if } \|U_i - U\| < \epsilon \text{ then } \|U^{\pi_i} - U\| < 2\epsilon/\gamma. \quad (17.9)$$

Dalam praktiknya, sering kali π_i menjadi optimal jauh sebelum U_i konvergen. Gambar 8.6 menunjukkan bagaimana galat maksimum dalam U_i dan kerugian kebijakan mendekati nol ketika proses iterasi nilai berlangsung untuk lingkungan 4×3 dengan $\gamma = 0,9$. Kebijakan π_i optimal ketika $i = 4$, meskipun galat maksimum dalam U_i masih 0,46.

Sekarang kita memiliki semua yang kita butuhkan untuk menggunakan iterasi nilai dalam praktik. Kita tahu bahwa iterasi tersebut konvergen ke utilitas yang tepat, kita dapat membatasi galat dalam estimasi utilitas jika kita berhenti setelah sejumlah iterasi yang terbatas, dan kita dapat membatasi kerugian kebijakan yang dihasilkan dari eksekusi kebijakan MEU yang sesuai. Sebagai catatan akhir, semua hasil di bagian ini bergantung pada diskonto dengan $\gamma < 1$. Jika $\gamma = 1$ dan lingkungan berisi status terminal, maka serangkaian hasil konvergensi dan batas galat yang serupa dapat diturunkan setiap kali kondisi teknis tertentu terpenuhi.

8.3 ITERASI KEBIJAKAN

Pada bagian sebelumnya, kita mengamati bahwa kebijakan optimal dapat diperoleh meskipun estimasi fungsi utilitas tidak akurat. Jika satu tindakan jelas lebih baik daripada semua tindakan lainnya, maka besaran utilitas yang tepat pada status yang terlibat tidak perlu tepat. Wawasan ini menyarankan cara alternatif untuk menemukan kebijakan optimal.

Algoritma iterasi kebijakan mengganti dua langkah berikut, dimulai dari beberapa kebijakan awal π_0 :

- Evaluasi kebijakan: dengan kebijakan π_i , hitung $U_i = U^{\pi_i}$, utilitas setiap status jika π_i dieksekusi.
- Perbaiki kebijakan: Hitung kebijakan MEU baru π_{i+1} , menggunakan one-step look-ahead berdasarkan U_i (seperti pada Persamaan (8.4)).

Algoritma berakhir ketika langkah perbaikan kebijakan tidak menghasilkan perubahan dalam utilitas. Pada titik ini, kita tahu bahwa fungsi utilitas U_i adalah titik tetap dari pembaruan Bellman, sehingga merupakan solusi untuk persamaan Bellman, dan π_i harus merupakan kebijakan optimal. Karena hanya ada kebijakan yang jumlahnya terbatas untuk ruang keadaan terbatas, dan setiap iterasi dapat ditunjukkan menghasilkan kebijakan yang lebih baik, iterasi kebijakan harus diakhiri. Algoritmenya ditunjukkan pada Gambar 8.7.

Langkah perbaikan kebijakan jelas mudah, tetapi bagaimana kita menerapkan rutinitas EVALUASI KEBIJAKAN? Ternyata melakukannya jauh lebih mudah daripada menyelesaikan persamaan Bellman standar (yang merupakan fungsi iterasi nilai), karena tindakan di setiap keadaan ditetapkan oleh kebijakan. Pada iterasi ke- i , kebijakan π_i menentukan tindakan $\pi_i(s)$ dalam keadaan s . Ini berarti bahwa kita memiliki versi persamaan Bellman (17.5) yang disederhanakan yang menghubungkan utilitas s (di bawah π_i) dengan utilitas tetangganya:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U_i(s'). \quad (8.10)$$



Gambar 8.6 Kesalahan maksimum $\|U_i - U\|$ dari estimasi utilitas dan kerugian kebijakan $\|U^{\pi_i} - U\|$, sebagai fungsi dari jumlah iterasi nilai iterasi.

Misalnya, misalkan π_i adalah kebijakan yang ditunjukkan pada Gambar 8.2(a). Maka kita memiliki $\pi_i(1, 1) = \text{Naik}$, $\pi_i(1, 2) = \text{Up}$, dan seterusnya, dan persamaan Bellman yang disederhanakan adalah

$$\begin{aligned} U_i(1,1) &= -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1), \\ U_i(1,2) &= -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2), \end{aligned}$$

⋮

Hal yang penting adalah bahwa persamaan ini bersifat linier, karena operator "maks" telah dihilangkan. Untuk n keadaan, kita memiliki n persamaan linier dengan n variabel yang tidak diketahui, yang dapat diselesaikan secara tepat dalam waktu $O(n^3)$ dengan metode aljabar linier standar.

Untuk ruang keadaan kecil, evaluasi kebijakan menggunakan metode solusi eksak sering kali merupakan pendekatan yang paling efisien. Untuk ruang keadaan besar, waktu $O(n^3)$ mungkin menjadi penghalang. Untungnya, tidak perlu melakukan evaluasi kebijakan eksak. Sebaliknya, kita dapat melakukan sejumlah langkah iterasi nilai yang disederhanakan (disederhanakan karena kebijakannya tetap) untuk memberikan perkiraan utilitas yang cukup baik. Pembaruan Bellman yang disederhanakan untuk proses ini adalah

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U_i(s'),$$

dan ini diulang k kali untuk menghasilkan estimasi utilitas berikutnya. Algoritma yang dihasilkan disebut iterasi kebijakan yang dimodifikasi. Algoritma ini seringkali jauh lebih efisien daripada iterasi kebijakan standar atau iterasi nilai.

```

function POLICY-ITERATION(mdp) returns a policy
  inputs: mdp, an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ 
  local variables:  $U$ , a vector of utilities for states in  $S$ , initially zero
                    $\pi$ , a policy vector indexed by state, initially random
  repeat
     $U \leftarrow$  POLICY-EVALUATION( $\pi, U, mdp$ )
    unchanged?  $\leftarrow$  true
    for each state  $s$  in  $S$  do
      if  $\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s']$  then do
         $\pi[s] \leftarrow \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$ 
      unchanged?  $\leftarrow$  false
  until unchanged?

```

Gambar 8.7 Algoritma iterasi kebijakan untuk menghitung kebijakan optimal.

Algoritma yang telah kami jelaskan sejauh ini memerlukan pembaruan utilitas atau kebijakan untuk semua status sekaligus. Ternyata hal ini tidak sepenuhnya diperlukan. Bahkan, pada setiap iterasi, kita dapat memilih subset status mana pun dan menerapkan salah satu jenis pembaruan (perbaikan kebijakan atau iterasi nilai yang disederhanakan) ke subset tersebut. Algoritma yang sangat umum ini disebut iterasi kebijakan asinkron. Dengan kondisi tertentu

pada kebijakan awal dan fungsi utilitas awal, iterasi kebijakan asinkron dijamin akan konvergen ke kebijakan optimal.

Kebebasan untuk memilih status apa pun yang akan dikerjakan berarti kita dapat merancang algoritma heuristik yang jauh lebih efisien—misalnya, algoritma yang berfokus pada pembaruan nilai status yang kemungkinan akan dicapai oleh kebijakan yang baik. Hal ini sangat masuk akal dalam kehidupan nyata: jika seseorang tidak berniat terjun dari tebing, ia seharusnya tidak menghabiskan waktu mengkhawatirkan nilai pasti dari status yang dihasilkan.

8.4 MDPS YANG DAPAT DIAMAT SEBAGIAN

Deskripsi proses keputusan Markov di Bagian 8.1 mengasumsikan bahwa lingkungan sepenuhnya dapat diamati. Dengan asumsi ini, agen selalu tahu keadaan di mana ia berada. Hal ini, dikombinasikan dengan asumsi Markov untuk model transisi, berarti bahwa kebijakan optimal hanya bergantung pada keadaan saat ini. Ketika lingkungan hanya dapat diamati sebagian, situasinya, bisa dikatakan, jauh kurang jelas.

Agen belum tentu tahu keadaan di mana ia berada, sehingga ia tidak dapat menjalankan tindakan $\pi(s)$ yang direkomendasikan untuk keadaan tersebut. Lebih lanjut, utilitas keadaan s dan tindakan optimal dalam s tidak hanya bergantung pada s , tetapi juga pada seberapa banyak agen tahu ketika ia berada dalam s . Karena alasan-alasan ini, MDP yang dapat diamati sebagian (atau POMDP—diucapkan "pom-dee-pees") biasanya dipandang jauh lebih sulit daripada MDP biasa. Namun, kita tidak dapat menghindari POMDP, karena dunia nyata adalah satu.

Definisi POMDP

Untuk memahami POMDP, pertama-tama kita harus mendefinisikannya dengan tepat. POMDP memiliki elemen yang sama dengan MDP—model transisi $P(s' | s, a)$, tindakan $A(s)$, dan fungsi imbalan $R(s)$ —tetapi, seperti masalah pencarian yang dapat diamati sebagian pada Bagian 4.4, POMDP juga memiliki model sensor $P(e | s)$. Di sini, seperti pada Bab 15, model sensor menentukan probabilitas mempersepsi bukti e dalam keadaan s .³ Sebagai contoh, kita dapat mengubah dunia 4×3 pada Gambar 8.1 menjadi POMDP dengan menambahkan sensor yang berisik atau parsial, alih-alih berasumsi bahwa agen mengetahui lokasinya secara tepat.

Sensor semacam itu dapat mengukur jumlah dinding yang berdekatan, yang kebetulan berjumlah 2 di semua kotak nonterminal kecuali yang ada di kolom ketiga, di mana nilainya adalah 1; versi yang berisik mungkin memberikan nilai yang salah dengan probabilitas 0,1. Kami mempelajari masalah perencanaan nondeterministik dan sebagian dapat diamati dan mengidentifikasi status keyakinan—himpunan status aktual yang mungkin dialami agen—sebagai konsep kunci untuk mendeskripsikan dan menghitung solusi. Dalam POMDP, status keyakinan b menjadi distribusi probabilitas atas semua status yang mungkin, seperti dalam Bab 15. Misalnya, status keyakinan awal untuk POMDP 4×3 dapat berupa distribusi seragam atas sembilan status nonterminal, yaitu, $\langle \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, 0, 0 \rangle$. Kami menulis $b(s)$ untuk probabilitas yang ditetapkan ke status aktual s oleh status keyakinan b . Agen dapat

menghitung status keyakinannya saat ini sebagai distribusi probabilitas bersyarat atas status aktual yang diberikan urutan persepsi dan tindakan sejauh ini.

Persamaan penyaringan rekursif dasar menunjukkan cara menghitung status keyakinan baru dari status keyakinan sebelumnya dan bukti baru. Untuk POMDP, kita juga memiliki tindakan yang perlu dipertimbangkan, tetapi hasilnya pada dasarnya sama. Jika $b(s)$ adalah status keyakinan sebelumnya, dan agen melakukan tindakan a dan kemudian mempersepsi bukti e , maka status keyakinan baru diberikan oleh

$$b'(s') = \alpha P(e | s') \sum_s P(s' | s, a) b(s),$$

di mana α adalah konstanta normalisasi yang membuat jumlah status kepercayaan menjadi 1. Dengan analogi dengan operator pembaruan untuk penyaringan, kita dapat menulis ini sebagai

$$b' = FORWARD(b, a, e). \quad (8.11)$$

Dalam POMDP 4×3 , misalkan agen bergerak ke Kiri dan sensornya melaporkan 1 dinding yang berdekatan; maka sangat mungkin (meskipun tidak dijamin, karena gerakan dan sensornya berisik) bahwa agen sekarang berada di (3,1).

Pemahaman mendasar yang diperlukan untuk memahami POMDP adalah ini: tindakan optimal hanya bergantung pada status keyakinan agen saat ini. Artinya, kebijakan optimal dapat dijelaskan dengan pemetaan $\pi^*(b)$ dari status keyakinan ke tindakan. Itu tidak bergantung pada status aktual agen tersebut. Ini adalah hal yang baik, karena agen tidak mengetahui status aktualnya; yang diketahuinya hanyalah status keyakinan. Oleh karena itu, siklus keputusan agen POMDP dapat dipecah menjadi tiga langkah berikut:

1. Diberikan status keyakinan saat ini b , jalankan tindakan $a = \pi^*(b)$.
2. Terima persepsi e .
3. Tetapkan status keyakinan saat ini ke $FORWARD(b, a, e)$ dan ulangi.

Sekarang kita dapat menganggap POMDP sebagai sesuatu yang memerlukan pencarian dalam ruang status keyakinan, sama seperti metode untuk masalah tanpa sensor dan kontingensi dalam Bab 4. Perbedaan utamanya adalah bahwa ruang status keyakinan POMDP bersifat kontinu, karena status keyakinan POMDP adalah distribusi probabilitas. Misalnya, status keyakinan untuk dunia 4×3 adalah titik dalam ruang kontinu 11 dimensi. Suatu tindakan mengubah status keyakinan, bukan hanya status fisik. Oleh karena itu, tindakan tersebut dievaluasi setidaknya sebagian menurut informasi yang diperoleh agen sebagai hasilnya. Oleh karena itu, POMDP menyertakan nilai informasi sebagai salah satu komponen masalah keputusan.

Mari kita lihat lebih cermat hasil tindakan. Secara khusus, mari kita hitung probabilitas bahwa agen dalam status keyakinan b mencapai status keyakinan b' setelah menjalankan tindakan a . Sekarang, jika kita mengetahui tindakan dan persepsi berikutnya, maka Persamaan

(8.11) akan memberikan pembaruan deterministik pada status keyakinan: $b' = FORWARD(b, a, e)$. Tentu saja, persepsi berikutnya belum diketahui, jadi agen mungkin tiba di salah satu dari beberapa kemungkinan status keyakinan b' , tergantung pada persepsi yang diterima. Probabilitas mempersepsi e , mengingat a dilakukan mulai dari status keyakinan b , diberikan dengan menjumlahkan semua status aktual s' yang mungkin dicapai agen:

$$\begin{aligned} P(e | a, b) &= \sum_{s'} P(e | a, s', b) P(s' | a, b) \\ &= \sum_{s'} P(e | s') P(s' | a, b) \\ &= \sum_{s'} P(e | a, s') \sum_s P(s' | s, a) b(s) . \end{aligned}$$

Mari kita tuliskan probabilitas mencapai b' dari b , dengan tindakan a , sebagai $P(b' | b, a)$. Maka kita akan memperoleh

$$\begin{aligned} P(b' | b, a) &= P(b' | a, b) = \sum_e P(b' | e, a, b) P(e | a, b) \\ &= \sum_e P(b' | e, a, b) \sum_{s'} P(e | s') \sum_s P(s' | s, a) b(s) , \end{aligned} \quad (8.12)$$

di mana $P(b' | e, a, b)$ bernilai 1 jika $b' = FORWARD(b, a, e)$ dan 0 jika tidak.

Persamaan (8.12) dapat dipandang sebagai pendefinisian model transisi untuk ruang keyakinan-keadaan. Kita juga dapat mendefinisikan fungsi imbalan untuk keadaan keyakinan (yaitu, imbalan yang diharapkan untuk keadaan aktual yang mungkin dialami agen):

$$p(b) = \sum_s b(s) R(s) .$$

Bersama-sama, $P(b' | b, a)$ dan $p(b)$ mendefinisikan MDP yang dapat diamati pada ruang status keyakinan. Lebih lanjut, dapat ditunjukkan bahwa kebijakan optimal untuk MDP ini, $\pi^*(b)$, juga merupakan kebijakan optimal untuk POMDP asli. Dengan kata lain, menyelesaikan POMDP pada ruang status fisik dapat direduksi menjadi menyelesaikan MDP pada ruang status keyakinan yang sesuai.

Fakta ini mungkin kurang mengejutkan jika kita ingat bahwa status keyakinan selalu dapat diamati oleh agen, menurut definisi. Perhatikan bahwa, meskipun kita telah mereduksi POMDP menjadi MDP, MDP yang kita peroleh memiliki ruang status kontinu (dan biasanya berdimensi tinggi). Tidak ada algoritma MDP yang berlaku langsung untuk MDP tersebut. Dua subbagian berikutnya menjelaskan algoritma iterasi nilai yang dirancang khusus untuk POMDP dan algoritma pengambilan keputusan daring.

Iterasi Nilai untuk POMDP

Bagian 8.2 menjelaskan algoritma iterasi nilai yang menghitung satu nilai utilitas untuk setiap keadaan. Dengan jumlah keadaan keyakinan yang tak terhingga, kita perlu lebih kreatif. Pertimbangkan kebijakan optimal π^* dan penerapannya dalam keadaan keyakinan spesifik b : kebijakan menghasilkan tindakan, kemudian, untuk setiap persepsi berikutnya, keadaan keyakinan diperbarui dan tindakan baru dihasilkan, dan seterusnya. Untuk b spesifik ini, oleh karena itu, kebijakan tersebut persis setara dengan rencana kondisional. Alih-alih memikirkan kebijakan, mari kita pikirkan rencana kondisional dan bagaimana utilitas yang diharapkan dari mengeksekusi rencana kondisional yang tetap bervariasi dengan keadaan keyakinan awal. Kami membuat dua pengamatan:

1. Misalkan utilitas pelaksanaan rencana kondisional tetap p yang dimulai dalam keadaan fisik s adalah $\alpha_p(s)$. Maka, utilitas ekspektasi pelaksanaan p dalam keadaan keyakinan b adalah $\sum_s b(s)\alpha_p(s)$, atau $b \cdot \alpha_p$ jika kita menganggap keduanya sebagai vektor. Oleh karena itu, utilitas ekspektasi dari rencana kondisional tetap bervariasi secara linear terhadap b ; artinya, utilitas tersebut berkorespondensi dengan hiperbidang dalam ruang keyakinan.
2. Pada keadaan keyakinan b mana pun, kebijakan optimal akan memilih untuk melaksanakan rencana kondisional dengan utilitas ekspektasi tertinggi; dan utilitas ekspektasi b dalam kebijakan optimal tersebut hanyalah utilitas dari rencana kondisional tersebut:

$$U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p.$$

Jika kebijakan optimal π^* memilih untuk mengeksekusi p mulai dari b , maka wajar untuk memperkirakan bahwa kebijakan tersebut mungkin memilih untuk mengeksekusi p dalam status keyakinan yang sangat dekat dengan b ; faktanya, jika kita membatasi kedalaman rencana kondisional, maka hanya ada banyak rencana seperti itu yang terbatas dan ruang kontinu status keyakinan umumnya akan terbagi menjadi beberapa wilayah, masing-masing berkorespondensi dengan rencana kondisional tertentu yang optimal di wilayah tersebut.

Dari kedua pengamatan ini, kita melihat bahwa fungsi utilitas $U(b)$ pada status keyakinan, yang merupakan nilai maksimum dari sekumpulan hiperbidang, akan bersifat linier sepotong-sepotong dan cembung.

Untuk mengilustrasikan hal ini, kita menggunakan dunia dua status sederhana. Status-status tersebut diberi label 0 dan 1, dengan $R(0) = 0$ dan $R(1) = 1$. Ada dua tindakan: Tetap di tempat dengan probabilitas 0,9 dan Pergi beralih ke status lain dengan probabilitas 0,9. Untuk saat ini, kita akan mengasumsikan faktor diskonto $\gamma = 1$. Sensor melaporkan status yang benar dengan probabilitas 0,6. Jelas, agen harus Tetap ketika ia mengira berada di keadaan 1 dan Pergi ketika ia mengira berada di keadaan 0.

Keuntungan dari dunia dua keadaan adalah bahwa ruang keyakinan dapat dipandang sebagai satu dimensi, karena kedua probabilitas harus berjumlah 1. Pada Gambar 8.8(a),

sumbu x merepresentasikan keadaan keyakinan, yang didefinisikan oleh $b(1)$, probabilitas berada di keadaan 1. Sekarang mari kita pertimbangkan rencana satu langkah $[Stay]$ dan $[Go]$, yang masing-masing menerima imbalan untuk keadaan saat ini diikuti oleh imbalan (yang didiskontokan) untuk keadaan yang dicapai setelah tindakan:

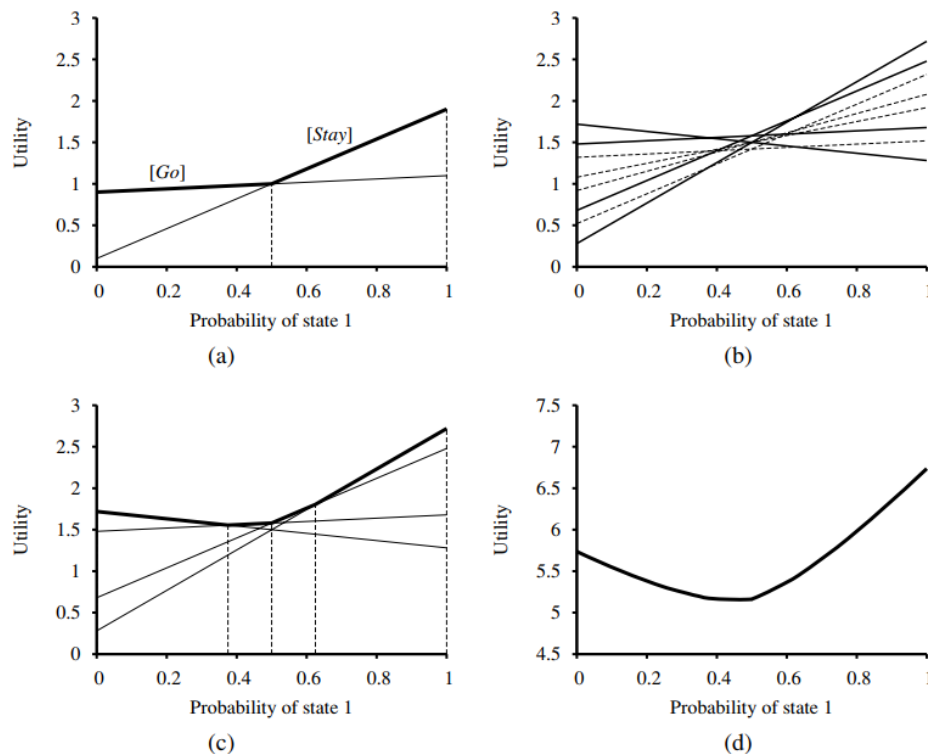
$$\alpha_{|Stay|}(0) = R(0) + \gamma(0.9R(0) + 0.1R(1)) = 0.1$$

$$\alpha_{|Stay|}(1) = R(1) + \gamma(0.9R(1) + 0.1R(0)) = 1.9$$

$$\alpha_{|Go|}(0) = R(0) + \gamma(0.9R(1) + 0.1R(0)) = 0.9$$

$$\alpha_{|Go|}(1) = R(1) + \gamma(0.9R(0) + 0.1R(1)) = 1.1$$

Bidang-bidang hiper (garis, dalam kasus ini) untuk $b \cdot \alpha_{|Stay|}$ dan $b \cdot \alpha_{|Go|}$ ditunjukkan pada Gambar 8.8(a) dan nilai maksimumnya ditunjukkan dengan huruf tebal. Oleh karena itu, garis tebal merepresentasikan fungsi utilitas untuk masalah horizon-hingga yang hanya memperbolehkan satu aksi, dan pada setiap "bagian" dari fungsi utilitas linear sepotong-sepotong, aksi optimal adalah aksi pertama dari rencana kondisional yang bersangkutan. Dalam kasus ini, kebijakan satu langkah optimal adalah $Stay$ ketika $b(1) > 0,5$ dan Go jika tidak.



Gambar 8.8 (a) Kegunaan dua rencana satu langkah sebagai fungsi dari keadaan keyakinan awal $b(1)$ untuk dunia dua-keadaan, dengan fungsi utilitas yang sesuai ditunjukkan dalam huruf tebal. (b) Utilitas untuk 8 rencana dua langkah yang berbeda. (c) Utilitas untuk empat rencana dua langkah yang tidak didominasi. (d) Fungsi utilitas untuk rencana delapan langkah yang optimal.

Setelah kita memiliki utilitas $\alpha_p(s)$ untuk semua rencana kondisional p dengan kedalaman 1 di setiap keadaan fisik s , kita dapat menghitung utilitas untuk rencana kondisional dengan kedalaman 2 dengan mempertimbangkan setiap kemungkinan aksi pertama, setiap kemungkinan persepsi berikutnya, dan kemudian setiap cara memilih rencana kedalaman-1 untuk dieksekusi untuk setiap persepsi:

[Stay; if Percept = 0 then Stay else Stay]
[Stay; if Percept = 0 then Stay else Go] ...

Terdapat delapan rencana kedalaman-2 yang berbeda secara keseluruhan, dan utilitasnya ditunjukkan pada Gambar 8.8(b). Perhatikan bahwa empat dari rencana tersebut, yang ditunjukkan sebagai garis putus-putus, bersifat suboptimal di seluruh ruang keyakinan—kita sebut saja rencana-rencana ini terdominasi, dan tidak perlu dipertimbangkan lebih lanjut. Terdapat empat rencana yang tidak terdominasi, yang masing-masing optimal di wilayah tertentu, seperti yang ditunjukkan pada Gambar 8.8(c). Wilayah-wilayah tersebut membagi ruang keadaan-keyakinan.

Kita ulangi proses untuk kedalaman 3, dan seterusnya. Secara umum, misalkan p adalah rencana kondisional kedalaman- d yang aksi awalnya adalah a dan subrencana kedalaman- $d - 1$ untuk persepsi e adalah $p.e$; maka

$$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} P(s' | s, a) \sum_e P(e | s') \alpha_{p.e}(s') \right) \quad (8.13)$$

Rekursi ini secara alami memberi kita algoritma iterasi nilai, yang digambarkan pada Gambar 8.9. Struktur algoritma dan analisis kesalahannya serupa dengan algoritma iterasi nilai dasar pada Gambar 8.4; perbedaan utamanya adalah, alih-alih menghitung satu angka utilitas untuk setiap keadaan, POMDP-VALUE-ITERATION mempertahankan kumpulan rencana tak terdominasi dengan hiperbidang utilitasnya. Kompleksitas algoritma terutama bergantung pada berapa banyak rencana yang dihasilkan.

Dengan $|A|$ aksi dan $|E|$ kemungkinan observasi, mudah untuk menunjukkan bahwa terdapat $|A|^{O(|E|d-1)}$ rencana kedalaman- d yang berbeda. Bahkan untuk dunia dua-keadaan yang sederhana dengan $d = 8$, jumlah pastinya adalah 2^{255} . Penghapusan rencana yang didominasi sangat penting untuk mengurangi pertumbuhan eksponensial ganda ini: jumlah rencana yang tidak didominasi dengan $d = 8$ hanya 144. Fungsi utilitas untuk 144 rencana ini ditunjukkan pada Gambar 8.8(d).

Perhatikan bahwa meskipun keadaan 0 memiliki utilitas yang lebih rendah daripada keadaan 1, keadaan keyakinan antara bahkan memiliki utilitas yang lebih rendah karena agen tidak memiliki informasi yang dibutuhkan untuk memilih tindakan yang baik. Inilah sebabnya mengapa informasi memiliki nilai dalam pengertian yang didefinisikan dalam Bagian 16.6 dan kebijakan optimal dalam POMDP sering kali mencakup tindakan pengumpulan informasi.

Dengan fungsi utilitas seperti itu, kebijakan yang dapat dieksekusi dapat diekstraksi dengan melihat hiperbidang mana yang optimal pada setiap keadaan keyakinan b dan mengeksekusi tindakan pertama dari rencana yang sesuai. Pada Gambar 8.8(d), kebijakan optimal yang sesuai masih sama dengan rencana kedalaman-1: *Stay* ketika $b(1) > 0,5$ dan *Go* sebaliknya.

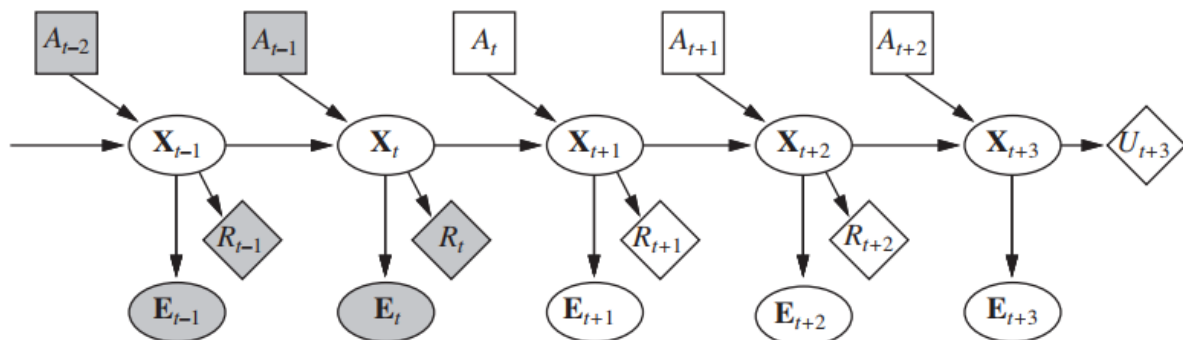
```

function POMDP-VALUE-ITERATION( $pomdp, \epsilon$ ) returns a utility function
  inputs:  $pomdp$ , a POMDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,
           sensor model  $P(e|s)$ , rewards  $R(s)$ , discount  $\gamma$ 
            $\epsilon$ , the maximum error allowed in the utility of any state
  local variables:  $U, U'$ , sets of plans  $p$  with associated utility vectors  $\alpha_p$ 

   $U' \leftarrow$  aset containing just the empty plan  $[],$  with  $\alpha_{[]} (s) = R(s)$ 
  repeat
     $U \leftarrow U'$ 
     $U' \leftarrow$  the set of all plans consisting of an action and, for each possible next
    percept,
      a plan in  $U$  with utility vectors computed according to Equation (17.13)
     $U' \leftarrow$  REMOVE-DOMINATED-PLANS( $U'$ )

```

Gambar 8.9 Sketsa tingkat tinggi algoritma iterasi nilai untuk POMDP. Langkah REMOVE-DOMINATED-PLANS dan uji MAX-DIFFERENCE biasanya diimplementasikan sebagai program linear.



Gambar 8.10 Struktur generik jaringan keputusan dinamis. Variabel dengan nilai yang diketahui diarsir. Waktu saat ini adalah t dan agen harus memutuskan apa yang harus dilakukan—yaitu, memilih nilai untuk A_t . Jaringan telah digulirkan ke masa depan selama tiga langkah dan merepresentasikan imbalan di masa mendatang, serta utilitas keadaan pada cakrawala pandang ke depan.

Dalam praktiknya, algoritme iterasi nilai pada Gambar 8.9 sangat tidak efisien untuk masalah yang lebih besar—bahkan POMDP 4×3 terlalu sulit. Alasan utamanya adalah, dengan n rencana bersyarat pada level d , algoritme tersebut membangun rencana bersyarat $|A| \cdot n^{|E|}$ pada level $d + 1$ sebelum menghilangkan yang didominasi. Sejak tahun 1970an, ketika

algoritme ini dikembangkan, telah ada beberapa kemajuan termasuk bentuk iterasi nilai yang lebih efisien dan berbagai jenis algoritme iterasi kebijakan.

Beberapa di antaranya dibahas dalam catatan di akhir bab. Namun, untuk POMDP umum, menemukan kebijakan optimal sangat sulit (PSPACE-sulit, sebenarnya—yaitu, sangat sulit). Masalah dengan beberapa lusin status sering kali tidak layak. Bagian berikutnya menjelaskan metode perkiraan yang berbeda untuk menyelesaikan POMDP, yang didasarkan pada pencarian lihat-ke-depan.

Agen Daring untuk POMDP

Di bagian ini, kami menguraikan pendekatan sederhana untuk desain agen untuk lingkungan stokastik yang dapat diamati sebagian. Elemen dasar desainnya sudah familier:

- Model transisi dan sensor direpresentasikan oleh jaringan Bayesian dinamis (DBN).
- Jaringan Bayesian dinamis diperluas dengan simpul keputusan dan utilitas, sebagaimana digunakan dalam jaringan keputusan. Model yang dihasilkan disebut jaringan keputusan dinamis, atau DDN.
- Algoritma penyaringan digunakan untuk menggabungkan setiap persepsi dan tindakan baru serta memperbarui representasi status keyakinan.
- Keputusan dibuat dengan memproyeksikan kemungkinan urutan tindakan ke depan dan memilih yang terbaik.

DBN adalah representasi terfaktor dalam terminologi Bab 2; DBN biasanya memiliki keunggulan kompleksitas eksponensial dibandingkan representasi atomik dan dapat memodelkan permasalahan dunia nyata yang cukup substansial. Oleh karena itu, desain agen merupakan implementasi praktis dari agen berbasis utilitas.

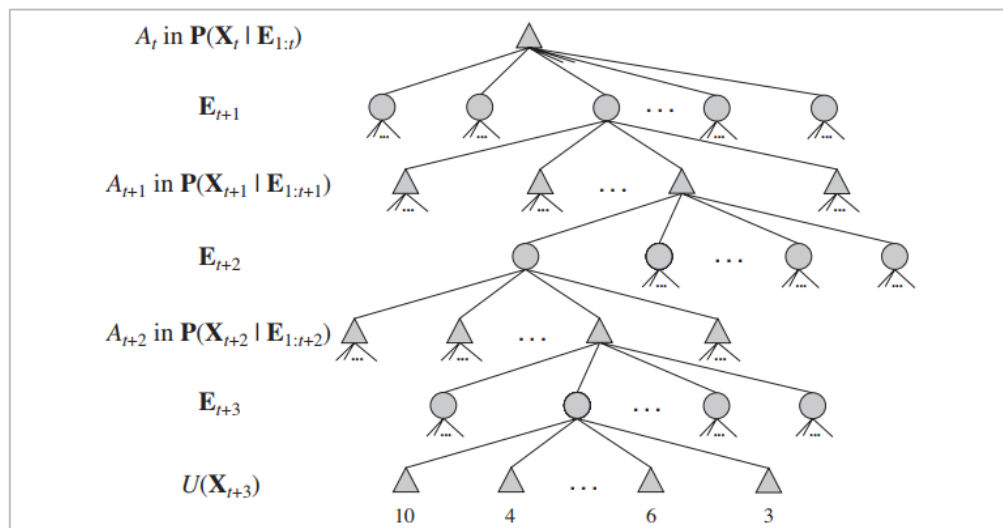
Dalam DBN, keadaan tunggal S_t menjadi sekumpulan variabel keadaan X_t , dan mungkin terdapat beberapa variabel bukti E_t . Kita akan menggunakan A_t untuk merujuk pada tindakan pada waktu t , sehingga model transisi menjadi $P(X_{t+1}|X_t, A_t)$ dan model sensor menjadi $P(E_t|X_t)$. Kita akan menggunakan R_t untuk merujuk pada imbalan yang diterima pada waktu t dan U_t untuk merujuk pada utilitas keadaan pada waktu t . (Keduanya merupakan variabel acak.) Dengan notasi ini, jaringan keputusan dinamis terlihat seperti yang ditunjukkan pada Gambar 8.10.

Jaringan keputusan dinamis dapat digunakan sebagai input untuk algoritma POMDP apa pun, termasuk algoritma untuk metode iterasi nilai dan kebijakan. Di bagian ini, kami fokus pada metode lihat-depan yang memproyeksikan urutan tindakan ke depan dari keadaan keyakinan saat ini dengan cara yang sama seperti yang dilakukan oleh algoritma permainan di Bab 5. Jaringan pada Gambar 8.10 telah diproyeksikan tiga langkah ke masa depan; keputusan saat ini dan masa depan A dan pengamatan masa depan E dan imbalan R semuanya tidak diketahui.

Perhatikan bahwa jaringan tersebut mencakup simpul untuk imbalan untuk X_{t+1} dan X_{t+2} , tetapi utilitas untuk X_{t+3} . Ini karena agen harus memaksimalkan jumlah (diskonto) dari semua imbalan masa depan, dan $U(X_{t+3})$ mewakili imbalan untuk X_{t+3} dan semua imbalan berikutnya. Seperti di Bab 5, kami berasumsi bahwa U hanya tersedia dalam beberapa bentuk perkiraan: jika nilai utilitas yang tepat tersedia, lihat-depan melampaui kedalaman 1 tidak

diperlukan. Gambar 8.11 menunjukkan bagian dari pohon pencarian yang sesuai dengan DDN lihat-depan tiga langkah pada Gambar 8.10.

Setiap simpul segitiga merupakan keadaan keyakinan di mana agen membuat keputusan A_{t+1} untuk $i = 0, 1, 2, \dots$. Simpul bulat (peluang) berkorespondensi dengan pilihan lingkungan, yaitu bukti apa yang datang E_{t+1} . Perhatikan bahwa tidak ada simpul peluang yang berkorespondensi dengan hasil tindakan; hal ini karena pembaruan keadaan keyakinan untuk suatu tindakan bersifat deterministik terlepas dari hasil aktualnya.



Gambar 8.11 Bagian dari solusi look-ahead DDN pada Gambar 8.10. Setiap keputusan akan diambil dalam status keyakinan yang ditunjukkan.

Keadaan keyakinan pada setiap simpul segitiga dapat dihitung dengan menerapkan algoritma penyaringan pada urutan persepsi dan tindakan yang mengarah padanya. Dengan cara ini, algoritma memperhitungkan fakta bahwa, untuk keputusan A_{t+1} , agen akan memiliki persepsi yang tersedia $E_{t+1}, \dots, E_{t+1}$, meskipun pada waktu t agen tidak mengetahui persepsi apa saja yang akan muncul. Dengan cara ini, agen teoretis-keputusan secara otomatis memperhitungkan nilai informasi dan akan menjalankan tindakan pengumpulan informasi jika diperlukan.

Keputusan dapat diekstraksi dari pohon pencarian dengan mencadangkan nilai utilitas dari daun, mengambil rata-rata pada simpul peluang, dan mengambil nilai maksimum pada simpul keputusan. Hal ini serupa dengan algoritma EXPECTIMINIMAX untuk pohon permainan dengan simpul peluang, kecuali bahwa (1) dapat juga terdapat imbalan pada keadaan non-daun dan (2) simpul keputusan sesuai dengan keadaan keyakinan, bukan keadaan aktual. Kompleksitas waktu pencarian menyeluruh hingga kedalaman d adalah $O(|A|^d \cdot |E|^d)$, dengan $|A|$ adalah jumlah tindakan yang tersedia dan $|E|$ adalah jumlah kemungkinan persepsi. (Perhatikan bahwa ini jauh lebih sedikit daripada jumlah rencana kondisional kedalaman- d yang dihasilkan oleh iterasi nilai.)

Untuk masalah di mana faktor diskonto γ tidak terlalu dekat dengan 1, pencarian dangkal seringkali cukup baik untuk memberikan keputusan yang mendekati optimal.

Dimungkinkan juga untuk mengaproksimasikan langkah rata-rata pada simpul peluang, dengan mengambil sampel dari himpunan kemungkinan persepsi alih-alih menjumlahkan semua kemungkinan persepsi. Ada berbagai cara lain untuk menemukan solusi aproksimasi yang baik dengan cepat.

Agen teoretis-keputusan berdasarkan jaringan keputusan dinamis memiliki sejumlah keunggulan dibandingkan dengan desain agen lain yang lebih sederhana yang disajikan di bab-bab sebelumnya. Secara khusus, mereka menangani lingkungan yang tidak pasti dan dapat dengan mudah merevisi "rencana" mereka untuk menangani bukti yang tidak terduga. Dengan model sensor yang tepat, mereka dapat menangani kegagalan sensor dan dapat merencanakan untuk mengumpulkan informasi.

Mereka menunjukkan "degradasi anggun" di bawah tekanan waktu dan dalam lingkungan yang kompleks, menggunakan berbagai teknik perkiraan. Jadi apa yang hilang? Salah satu cacat dari algoritma berbasis DDN kami adalah ketergantungannya pada pencarian maju melalui ruang keadaan, daripada menggunakan teknik perencanaan hierarkis dan teknik perencanaan lanjutan lainnya. Ada upaya untuk memperluas teknik ini ke domain probabilistik, tetapi sejauh ini terbukti tidak efisien.

Masalah kedua yang terkait adalah sifat bahasa DDN yang pada dasarnya proposisional. Kami ingin dapat memperluas beberapa ide untuk bahasa probabilistik orde pertama ke masalah pengambilan keputusan. Penelitian terkini telah menunjukkan bahwa perluasan ini memungkinkan dan memiliki manfaat yang signifikan, seperti yang dibahas dalam catatan di akhir bab.

8.5 KEPUTUSAN DENGAN BANYAK AGEN: TEORI PERMAINAN

Bab ini berfokus pada pengambilan keputusan dalam lingkungan yang tidak pasti. Namun, bagaimana jika ketidakpastian tersebut disebabkan oleh agen lain dan keputusan yang mereka buat? Dan bagaimana jika keputusan agen tersebut pada gilirannya dipengaruhi oleh keputusan kita? Namun, di sana, kita terutama membahas permainan giliran dalam lingkungan yang sepenuhnya dapat diamati, yang untuknya pencarian minimax dapat digunakan untuk menemukan langkah optimal. Di bagian ini, kita mempelajari aspek-aspek teori permainan yang menganalisis permainan dengan langkah simultan dan sumber-sumber lain dari observabilitas parsial. (Para ahli teori permainan menggunakan istilah informasi sempurna dan informasi tidak sempurna, alih-alih dapat diamati sepenuhnya dan sebagian.) Teori permainan dapat digunakan setidaknya dalam dua cara:

1. Desain agen: Teori permainan dapat menganalisis keputusan agen dan menghitung utilitas yang diharapkan untuk setiap keputusan (dengan asumsi bahwa agen lain bertindak optimal menurut teori permainan). Misalnya, dalam permainan Morra dua jari, dua pemain, O dan E , secara bersamaan menunjukkan satu atau dua jari. Misalkan jumlah total jari adalah f . Jika f ganjil, O mengumpulkan f rupiah dari E ; dan jika f genap, E mengumpulkan f rupiah dari O . Teori permainan dapat menentukan strategi terbaik melawan pemain rasional dan hasil yang diharapkan untuk setiap pemain.

2. Perancangan mekanisme: Ketika suatu lingkungan dihuni oleh banyak agen, aturan lingkungan tersebut (yaitu, permainan yang harus dimainkan oleh agen) dapat ditentukan sehingga kebaikan kolektif semua agen dapat dimaksimalkan ketika setiap agen mengadopsi solusi teori permainan yang memaksimalkan utilitasnya sendiri. Misalnya, teori permainan dapat membantu merancang protokol untuk sekumpulan router lalu lintas internet sehingga setiap router memiliki insentif untuk bertindak sedemikian rupa sehingga throughput global dapat dimaksimalkan. Perancangan mekanisme juga dapat digunakan untuk membangun sistem multiagen cerdas yang memecahkan masalah kompleks secara terdistribusi.

Permainan Satu Langkah

Kita mulai dengan mempertimbangkan serangkaian permainan terbatas: permainan di mana semua pemain mengambil tindakan secara bersamaan dan hasil permainan didasarkan pada serangkaian tindakan tunggal ini. (Sebenarnya, tidak penting bahwa tindakan tersebut terjadi pada waktu yang sama persis; yang penting adalah tidak ada pemain yang mengetahui pilihan pemain lain.) Pembatasan pada satu langkah (dan penggunaan kata "permainan") mungkin membuat hal ini tampak sepele, tetapi sebenarnya, teori permainan adalah bisnis yang serius.

Teori ini digunakan dalam situasi pengambilan keputusan termasuk pelelangan hak pengeboran minyak dan hak spektrum frekuensi nirkabel, proses kebangkrutan, pengembangan produk dan keputusan penetapan harga, dan pertahanan nasional—situasi yang melibatkan miliaran dolar dan ratusan ribu nyawa. Permainan satu langkah didefinisikan oleh tiga komponen:

- Pemain atau agen yang akan membuat keputusan. Permainan dua pemain telah menerima perhatian paling besar, meskipun permainan n -pemain untuk $n > 2$ juga umum. Kami memberi pemain nama dengan huruf kapital, seperti Alice dan Bob atau O dan E .
- Tindakan yang dapat dipilih pemain. Kami akan memberi tindakan nama dengan huruf kecil, seperti one atau testify. Pemain mungkin memiliki atau tidak memiliki serangkaian tindakan yang sama.
- Fungsi pembayaran yang memberikan utilitas kepada setiap pemain untuk setiap kombinasi tindakan oleh semua pemain. Untuk permainan satu langkah, fungsi pembayaran dapat direpresentasikan oleh matriks, representasi yang dikenal sebagai bentuk strategis (juga disebut bentuk normal). Matriks pembayaran untuk Morra dua jari adalah sebagai berikut:

	$O: \text{satu}$	$O: \text{dua}$
$E: \text{satu}$	$E = +2, O = -2$	$E = -3, O = +3$
$E: \text{dua}$	$E = -3, O = +3$	$E = +4, O = -4$

Misalnya, pojok kanan bawah menunjukkan bahwa ketika pemain O memilih aksi dua dan E juga memilih dua, hasil akhirnya adalah +4 untuk E dan -4 untuk O .

Setiap pemain dalam permainan harus mengadopsi dan kemudian mengeksekusi sebuah strategi (yang merupakan nama yang digunakan dalam teori permainan untuk sebuah kebijakan). Strategi murni adalah kebijakan deterministik; untuk permainan satu langkah, strategi murni hanyalah sebuah aksi tunggal.

Untuk banyak permainan, seorang agen dapat bermain lebih baik dengan strategi campuran, yang merupakan kebijakan acak yang memilih aksi berdasarkan distribusi probabilitas. Strategi campuran yang memilih aksi a dengan probabilitas p dan aksi b sebaliknya ditulis $[p: a; (1 - p): b]$. Misalnya, strategi campuran untuk Morra dua jari mungkin adalah $[0,5: \text{satu}; 0,5: \text{dua}]$. Profil strategi adalah penugasan strategi kepada setiap pemain; dengan profil strategi tersebut, hasil permainan adalah nilai numerik untuk setiap pemain.

1. Beberapa definisi MDP memungkinkan imbalan bergantung pada tindakan dan hasil juga, sehingga fungsi imbalannya adalah $R(s, a, s^*)$. Hal ini menyederhanakan deskripsi beberapa lingkungan tetapi tidak mengubah masalah secara mendasar.
2. Meskipun hal ini tampak jelas, hal ini tidak berlaku untuk kebijakan berhorizon terbatas atau cara lain untuk menggabungkan imbalan dari waktu ke waktu. Pembuktiannya mengikuti langsung dari keunikan fungsi utilitas pada keadaan.
3. Seperti halnya fungsi imbalan untuk MDP, model sensor juga dapat bergantung pada status tindakan dan hasil, tetapi sekali lagi, perubahan ini tidak fundamental.
4. Morra adalah versi rekreasi dari permainan inspeksi. Dalam permainan semacam itu, seorang inspektur memilih hari untuk memeriksa fasilitas (seperti restoran atau pabrik senjata biologis), dan operator fasilitas memilih hari untuk menyembunyikan semua barang berbahaya. Inspektur menang jika harinya berbeda, dan operator fasilitas menang jika harinya sama.

Solusi untuk sebuah permainan adalah profil strategi di mana setiap pemain mengadopsi strategi rasional. Kita akan melihat bahwa isu terpenting dalam teori permainan adalah mendefinisikan apa arti "rasional" ketika setiap agen hanya memilih sebagian dari profil strategi yang menentukan hasilnya. Penting untuk dipahami bahwa hasil adalah hasil aktual dari permainan, sementara solusi adalah konstruksi teoretis yang digunakan untuk menganalisis permainan. Kita akan melihat bahwa beberapa permainan hanya memiliki solusi dalam strategi campuran. Namun, itu tidak berarti bahwa seorang pemain harus benar-benar mengadopsi strategi campuran untuk menjadi rasional.

Perhatikan kisah berikut: Dua tersangka pencuri, Alice dan Bob, tertangkap basah di dekat lokasi pencurian dan diinterogasi secara terpisah. Seorang jaksa menawarkan kesepakatan kepada masing-masing: jika Anda bersaksi melawan rekan Anda sebagai pemimpin komplotan pencurian, Anda akan bebas karena kooperatif, sementara rekan Anda akan menjalani hukuman 10 tahun penjara. Namun, jika Anda berdua bersaksi melawan satu sama lain, Anda berdua akan dihukum 5 tahun. Alice dan Bob juga tahu bahwa jika keduanya menolak bersaksi, mereka hanya akan menjalani hukuman 1 tahun penjara untuk dakwaan yang lebih ringan, yaitu memiliki barang curian. Kini Alice dan Bob menghadapi apa yang disebut **dilema tahanan**: haruskah mereka bersaksi atau menolak? Sebagai agen rasional,

Alice dan Bob masing-masing ingin memaksimalkan utilitas yang diharapkan. Mari kita asumsikan Alice sama sekali tidak peduli dengan nasib pasangannya, sehingga utilitasnya berkurang sebanding dengan jumlah tahun yang akan ia habiskan di penjara, terlepas dari apa yang terjadi pada Bob. Bob merasakan hal yang persis sama. Untuk membantu mencapai keputusan rasional, mereka berdua menyusun matriks imbalan berikut:

	Alice: testify	Alice: refuse
Bob: testify	A = -5, B = -5	A = -10, B = 0
Bob: refuse	A = 0, B = -10	A = -1, B = -1

Alice menganalisis matriks hasil sebagai berikut: "Misalkan Bob bersaksi. Maka saya mendapatkan 5 tahun jika saya bersaksi dan 10 tahun jika tidak, jadi dalam kasus itu bersaksi lebih baik. Di sisi lain, jika Bob menolak, maka saya mendapatkan 0 tahun jika saya bersaksi dan 1 tahun jika saya menolak, jadi dalam kasus itu juga bersaksi lebih baik. Jadi dalam kedua kasus, lebih baik bagi saya untuk bersaksi, jadi itulah yang harus saya lakukan."

Alice telah menemukan bahwa *bersaksi* adalah **strategi dominan** untuk permainan tersebut. Kita katakan bahwa strategi s untuk pemain p **sangat mendominasi** strategi s' jika hasil untuk s lebih baik untuk p daripada hasil untuk s' , untuk setiap pilihan strategi oleh pemain lain. Strategi s **secara lemah mendominasi** s' jika s lebih baik daripada s' pada setidaknya satu profil strategi dan tidak lebih buruk pada yang lain. Strategi dominan adalah strategi yang mendominasi semua strategi lainnya. Memainkan strategi yang didominasi adalah irasional, dan tidak rasional jika tidak memainkan strategi dominan jika ada. Karena rasional, Alice memilih strategi dominan. Kita hanya perlu sedikit terminologi lagi: kita katakan bahwa suatu hasil adalah **Pareto optimal**⁵ jika tidak ada hasil lain yang diinginkan semua pemain. Suatu hasil adalah **Pareto didominasi** oleh hasil lain jika semua pemain menginginkan hasil yang lain.

Jika Alice cerdas sekaligus rasional, ia akan terus bernalar sebagai berikut: Strategi dominan Bob juga untuk bersaksi. Oleh karena itu, ia akan bersaksi dan kita berdua akan mendapatkan hukuman lima tahun. Ketika setiap pemain memiliki strategi dominan, kombinasi strategi-strategi tersebut disebut **ekuilibrium strategi dominan**. Secara umum, profil strategi membentuk **ekuilibrium** jika tidak ada pemain yang dapat memperoleh keuntungan dengan berganti strategi, mengingat setiap pemain lain tetap menggunakan strategi yang sama. Ekuilibrium pada dasarnya adalah **optimum lokal** dalam ruang kebijakan; ekuilibrium adalah puncak dari sebuah puncak yang menurun di sepanjang setiap dimensi, di mana sebuah dimensi berkorespondensi dengan pilihan strategi seorang pemain.

Matematikawan John Nash (1928–) membuktikan bahwa *setiap permainan memiliki setidaknya satu ekuilibrium*. Konsep umum ekuilibrium sekarang disebut **ekuilibrium Nash** untuk menghormatinya. Jelas, ekuilibrium strategi dominan adalah ekuilibrium, tetapi beberapa permainan memiliki ekuilibrium Nash tetapi tidak memiliki strategi dominan.

Dilema dalam dilema tahanan adalah bahwa hasil ekuilibrium lebih buruk bagi kedua pemain daripada hasil yang akan mereka dapatkan jika mereka berdua menolak untuk

bersaksi. Dengan kata lain, *(testify, testify)* didominasi Pareto oleh hasil $(-1, -1)$ dari *(refuse, refuse)*. Adakah cara bagi Alice dan Bob untuk mencapai hasil $(-1, -1)$? Tentu saja merupakan pilihan yang *diperbolehkan* bagi keduanya untuk menolak bersaksi, tetapi sulit untuk melihat bagaimana agen rasional dapat mencapainya, mengingat definisi permainannya. Setiap pemain yang mempertimbangkan untuk bermain *menolak* akan menyadari bahwa ia akan bermain *bersaksi* dengan lebih baik. Itulah daya tarik dari sebuah titik ekuilibrium. Para ahli teori permainan sepakat bahwa ekuilibrium Nash merupakan syarat perlu untuk menjadi solusi—meskipun mereka tidak sependapat apakah itu merupakan syarat cukup.

Cukup mudah untuk mencapai solusi *(refuse, refuse)* jika kita memodifikasi permainannya. Misalnya, kita dapat mengubahnya menjadi **permainan berulang** di mana para pemain tahu bahwa mereka akan bertemu lagi. Atau para agen mungkin memiliki keyakinan moral yang mendorong kerja sama dan keadilan. Itu berarti mereka memiliki fungsi utilitas yang berbeda, yang membutuhkan matriks hasil yang berbeda, menjadikannya permainan yang berbeda. Kita akan melihat nanti bahwa agen dengan daya komputasi terbatas, alih-alih kemampuan untuk bernalar secara rasional mutlak, dapat mencapai hasil non-ekuilibrium, seperti halnya agen yang tahu bahwa agen lain memiliki rasionalitas terbatas. Dalam setiap kasus, kita mempertimbangkan permainan yang berbeda dari yang dijelaskan oleh matriks hasil di atas.

Sekarang mari kita lihat permainan yang tidak memiliki strategi dominan. Acme, produsen konsol permainan video, harus memutuskan apakah mesin permainan berikutnya akan menggunakan cakram Blu-ray atau DVD. Sementara itu, produsen perangkat lunak gim video Best perlu memutuskan apakah akan memproduksi gim berikutnya dalam format Blu-ray atau DVD. Keuntungan bagi keduanya akan positif jika sesuai dan negatif jika tidak sesuai, seperti yang ditunjukkan dalam matriks pembayaran berikut:

	Acme: bluray	Acme: dvd
Best: bluray	A = +9, B = +9	A = -4, B = -1
Best: dvd	A = -3, B = -1	A = +5, B = +5

Tidak ada keseimbangan strategi dominan untuk permainan ini, tetapi ada dua keseimbangan Nash: *(bluray, bluray)* dan *(dvd, dvd)*. Kita tahu ini adalah keseimbangan Nash karena jika salah satu pemain secara sepihak pindah ke strategi yang berbeda, pemain itu akan dirugikan. Sekarang para agen memiliki masalah: *ada beberapa solusi yang dapat diterima, tetapi jika setiap agen mengincar solusi yang berbeda, maka kedua agen akan menderita*. Bagaimana mereka bisa menyepakati suatu solusi? Satu jawaban adalah bahwa keduanya harus memilih solusi Pareto-optimal *(bluray, bluray)*; yaitu, kita dapat membatasi definisi "solusi" pada keseimbangan Nash Pareto-optimal yang unik *asalkan ada*. Setiap permainan memiliki setidaknya satu solusi Pareto-optimal, tetapi sebuah permainan mungkin memiliki beberapa, atau mungkin bukan titik keseimbangan. Misalnya, jika *(bluray, bluray)* memiliki hasil $(5, 5)$, maka akan ada dua titik keseimbangan Pareto-optimal yang sama. Untuk memilih di antara keduanya, para agen dapat menebak atau *berkomunikasi*, yang dapat dilakukan dengan

menetapkan konvensi yang mengurutkan solusi sebelum permainan dimulai atau dengan bernegosiasi untuk mencapai solusi yang saling menguntungkan selama permainan (yang berarti memasukkan tindakan komunikatif sebagai bagian dari permainan sekuensial). Komunikasi dengan demikian muncul dalam teori permainan karena alasan yang persis sama dengan yang muncul dalam perencanaan multiagen di Bagian 8.4. Permainan di mana pemain perlu berkomunikasi seperti ini disebut **permainan koordinasi**.

Sebuah permainan dapat memiliki lebih dari satu ekuilibrium Nash; bagaimana kita tahu bahwa setiap permainan harus memiliki setidaknya satu? Beberapa permainan tidak memiliki ekuilibrium Nash *strategi murni*. Pertimbangkan, misalnya, profil strategi murni apa pun untuk Morra dua jari. Jika jumlah total jari genap, maka O akan ingin beralih; di sisi lain (bisa dikatakan), jika totalnya ganjil, maka E akan ingin beralih. Oleh karena itu, tidak ada profil strategi murni yang dapat menjadi ekuilibrium dan kita harus mencari strategi campuran sebagai gantinya.

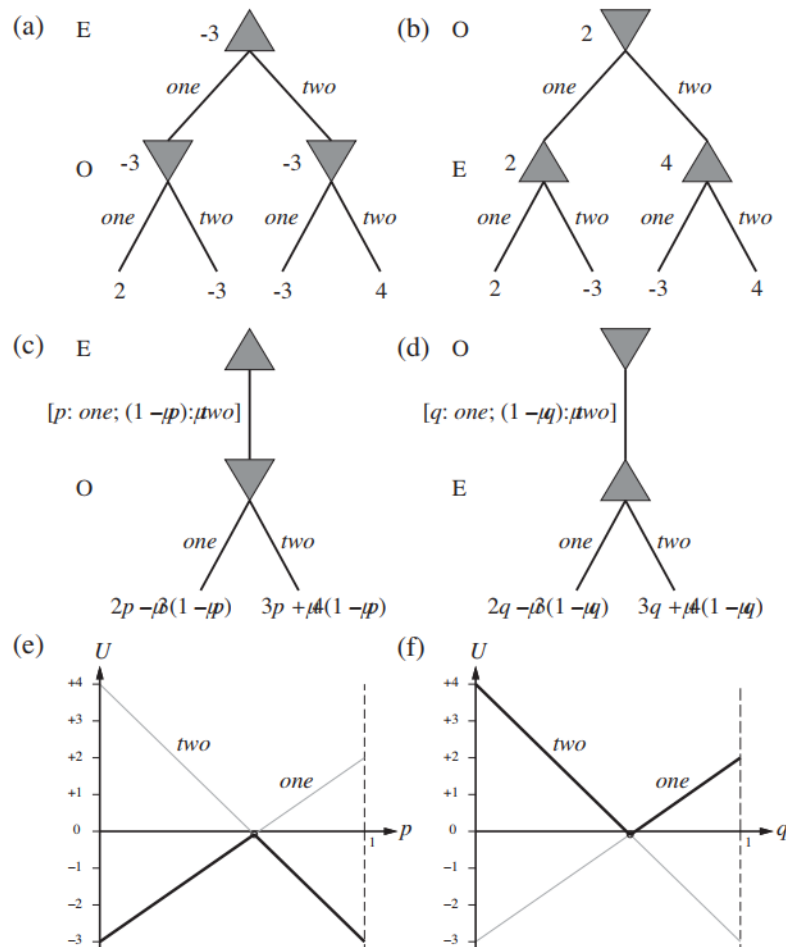
Tetapi strategi campuran yang mana? Pada tahun 1928, von Neumann mengembangkan metode untuk menemukan strategi campuran *optimal* untuk **permainan zero-sum** dua pemain—permainan di mana jumlah hasil selalu nol.⁶ Jelas, Morra adalah salah satu contoh permainan tersebut. Untuk permainan zero-sum dua pemain, kita tahu bahwa hasil yang diperoleh sama dan berlawanan arah, jadi kita perlu mempertimbangkan hasil dari hanya satu pemain, yang akan menjadi pemaksimal (seperti di Bab 5). Untuk Morra, kita memilih pemain genap E sebagai pemaksimal, sehingga kita dapat mendefinisikan matriks hasil dengan nilai $U_E(e, o)$ —hasil untuk E jika E melakukan e dan O melakukan o . (Untuk kenyamanan, kita menyebut pemain E "dia" dan O "dia.") Metode von Neumann disebut teknik **maximin**, dan cara kerjanya sebagai berikut:

- Misalkan kita mengubah aturan sebagai berikut: pertama E memilih strateginya dan mengungkapkannya kepada O . Kemudian O memilih strateginya, dengan mengetahui strategi E . Akhirnya, kita mengevaluasi hasil yang diharapkan dari permainan berdasarkan strategi yang dipilih. Ini memberi kita permainan bergiliran yang dapat kita terapkan algoritma **minimax** standar dari Bab 5. Misalkan ini memberikan hasil $U_{E,O}$. Jelas, permainan ini menguntungkan O , sehingga utilitas sejati U dari permainan asli (dari sudut pandang E) setidaknya $U_{E,O}$. Misalnya, jika kita hanya melihat strategi murni, pohon permainan minimax memiliki nilai akar -3 (lihat Gambar 8.12(a)), sehingga kita tahu bahwa $U \geq -3$.
- Sekarang misalkan kita mengubah aturan untuk memaksa O mengungkapkan strateginya terlebih dahulu, diikuti oleh E . Maka nilai minimax dari permainan ini adalah $U_{O,E}$ dan karena permainan ini menguntungkan E , kita tahu bahwa U paling banyak adalah $U_{E,O}$. Dengan strategi murni, nilainya adalah $+2$ (lihat Gambar 8.12(b)), sehingga kita tahu $U \leq +2$.

Dengan menggabungkan kedua argumen ini, kita melihat bahwa utilitas sejati U dari solusi permainan asli harus memenuhi

$$U_{E,O} \leq U \leq U_{O,E} \quad \text{Atau dalam kasus ini} \quad -3 \leq U \leq 2$$

Untuk menentukan nilai U , kita perlu mengalihkan analisis kita ke strategi campuran. Pertama, amati hal berikut: *setelah pemain pertama mengungkapkan strateginya, pemain kedua mungkin juga memilih strategi murni*. Alasannya sederhana: jika pemain kedua memainkan strategi campuran, $[p: \text{one}; (1-p): \text{two}]$, utilitas yang diharapkan adalah kombinasi linear $(p \cdot u_{\text{one}} + (1-p) \cdot u_{\text{two}})$ dari utilitas strategi murni, u_{one} dan u_{two} . Kombinasi linear ini tidak akan pernah lebih baik daripada yang lebih baik dari u_{one} dan u_{two} , jadi pemain kedua dapat memilih yang lebih baik.



Gambar 8.12 (a) dan (b): Pohon permainan Minimax untuk Morra dua jari jika para pemain bergiliran memainkan strategi murni. (c) dan (d): Pohon permainan berparameter di mana pemain pertama memainkan strategi campuran. Hasil bergantung pada parameter probabilitas (p atau q) dalam strategi campuran. (e) dan (f): Untuk setiap nilai parameter probabilitas tertentu, pemain kedua akan memilih tindakan yang "lebih baik" dari kedua tindakan tersebut, sehingga nilai strategi campuran pemain pertama ditentukan oleh garis tebal. Pemain pertama akan memilih parameter probabilitas untuk strategi campuran di titik potong.

Dengan mengingat pengamatan ini, pohon minimax dapat dianggap memiliki cabang tak terhingga banyaknya di akar, yang sesuai dengan strategi campuran tak terhingga

banyaknya yang dapat dipilih pemain pertama. Masing-masing mengarah ke simpul dengan dua cabang yang sesuai dengan strategi murni untuk pemain kedua. Kita dapat menggambarkan pohon-pohon tak terhingga ini secara terhingga dengan memiliki satu pilihan "parameter" di akarnya:

- Jika E memilih pertama, situasinya seperti yang ditunjukkan pada Gambar 8.12(c). E memilih strategi $[p: \text{one}; (1-p): \text{two}]$ di akar, dan kemudian O memilih strategi murni (dan karenanya sebuah langkah) dengan nilai p . Jika O memilih *satu*, hasil yang diharapkan (bagi E) adalah $2p - 3(1-p) = 5p - 3$; jika O memilih *dua*, hasil yang diharapkan adalah $-3p + 4(1-p) = 4 - 7p$. Kita dapat menggambar kedua hasil ini sebagai garis lurus pada grafik, di mana p berkisar dari 0 hingga 1 pada sumbu x , seperti yang ditunjukkan pada Gambar 8.12(e). O , si peminimal, akan selalu memilih garis yang lebih rendah dari kedua garis tersebut, seperti yang ditunjukkan oleh garis-garis tebal pada gambar. Oleh karena itu, hal terbaik yang dapat dilakukan E pada akhirnya adalah memilih p untuk berada di titik perpotongan, yaitu di mana

$$5p - 3 = 4 - 7p \Rightarrow p = 7/12 .$$

Utilitas untuk E pada titik ini adalah $U_{E,O} = -1/12$.

- Jika O bergerak lebih dulu, situasinya seperti yang ditunjukkan pada Gambar 8.12(d). O memilih strategi $[q: \text{one}; (1-q): \text{two}]$ di akar, lalu E memilih langkah berdasarkan nilai q . Hasil akhirnya adalah $2q - 3(1-q) = 5q - 3$ dan $-3q + 4(1-q) = 4 - 7q$.⁷ Sekali lagi, Gambar 8.12(f) menunjukkan bahwa hal terbaik yang dapat dilakukan O di akar adalah memilih titik potong:

$$5q - 3 = 4 - 7q \Rightarrow q = 7/12 .$$

Utilitas untuk E pada titik ini adalah $U_{E,O} = -1/12$

Sekarang kita tahu bahwa utilitas sebenarnya dari permainan asli terletak di antara $-1/12$ dan $-1/12$, yaitu, tepat $-1/12$! (Moralnya adalah lebih baik menjadi O daripada E jika Anda memainkan permainan ini.) Lebih lanjut, utilitas sebenarnya dicapai dengan strategi campuran $[7/12: \text{one}; 5/12: \text{two}]$, yang harus dimainkan oleh kedua pemain. Strategi ini disebut **ekuilibrium maksimin** permainan, dan merupakan ekuilibrium Nash. Perhatikan bahwa setiap strategi komponen dalam strategi campuran ekuilibrium memiliki utilitas yang diharapkan sama. Dalam hal ini, baik *satu* maupun *dua* memiliki utilitas yang diharapkan sama, $-1/12$, dengan strategi campuran itu sendiri.

Hasil kami untuk Morra dua jari adalah contoh dari hasil umum von Neumann: *setiap permainan zero-sum dua pemain memiliki ekuilibrium maksimin ketika Anda mengizinkan strategi campuran*. Lebih lanjut, setiap ekuilibrium Nash dalam permainan zero-sum merupakan maksimin bagi kedua pemain. Seorang pemain yang mengadopsi strategi maksimin memiliki dua jaminan: Pertama, tidak ada strategi lain yang dapat bekerja lebih baik melawan lawan yang bermain baik (meskipun beberapa strategi lain mungkin lebih baik dalam

mengeksploitasi lawan yang melakukan kesalahan irasional). Kedua, pemain tetap bermain sama baiknya meskipun strategi tersebut diungkapkan kepada lawan.

Algoritma umum untuk menemukan ekuilibrium maksimin dalam permainan zero-sum agak lebih rumit daripada yang mungkin ditunjukkan pada Gambar 8.12(e) dan (f). Ketika terdapat n kemungkinan tindakan, strategi campuran adalah sebuah titik dalam ruang n -dimensi dan garis-garisnya menjadi bidang-hiper. Ada juga kemungkinan beberapa strategi murni untuk pemain kedua didominasi oleh strategi lain, sehingga strategi tersebut tidak optimal terhadap strategi *apa pun* untuk pemain pertama. Setelah menghilangkan semua strategi tersebut (yang mungkin harus dilakukan berulang kali), pilihan optimal pada akhirnya adalah titik potong tertinggi (atau terendah) dari bidang-hiper yang tersisa. Menemukan pilihan ini merupakan contoh masalah **pemrograman linear**: memaksimalkan fungsi objektif yang tunduk pada batasan linear. Masalah semacam ini dapat diselesaikan dengan teknik standar dalam polinomial waktu dalam jumlah tindakan (dan dalam jumlah bit yang digunakan untuk menentukan fungsi imbalan, jika Anda ingin lebih teknis).

Pertanyaannya tetap, apa yang seharusnya *dilakukan* agen rasional dalam memainkan satu permainan Morra? Agen rasional akan menyimpulkan fakta bahwa [7/12: *one*; 5/12: *two*] adalah strategi ekuilibrium maksimin, dan akan berasumsi bahwa ini adalah pengetahuan bersama dengan lawan rasional. Agen dapat menggunakan dadu 12 sisi atau generator bilangan acak untuk memilih secara acak sesuai dengan strategi campuran ini, yang dalam hal ini hasil yang diharapkan adalah $-1/12$ untuk E . Atau agen dapat memutuskan untuk memainkan *satu*, atau *dua*. Dalam kedua kasus tersebut, hasil yang diharapkan tetap $-1/12$ untuk E . Anehnya, memilih tindakan tertentu secara sepihak tidak memengaruhi hasil yang diharapkan, tetapi membiarkan agen lain mengetahui bahwa ia telah *membuat* keputusan sepihak tersebut memengaruhi hasil yang diharapkan, karena lawan dapat menyesuaikan strateginya.

Menemukan ekuilibria dalam permainan non-zero-sum agak lebih rumit. Pendekatan umum memiliki dua langkah: (1) Hitung semua kemungkinan subset tindakan yang dapat membentuk strategi campuran. Misalnya, pertama-tama coba semua profil strategi di mana setiap pemain menggunakan satu tindakan, kemudian profil di mana setiap pemain menggunakan satu atau dua tindakan, dan seterusnya.

Ini eksponensial dalam jumlah tindakan, sehingga hanya berlaku untuk permainan yang relatif kecil. (2) Untuk setiap profil strategi yang disebutkan dalam (1), periksa apakah itu ekuilibrium. Ini dilakukan dengan menyelesaikan serangkaian persamaan dan pertidaksamaan yang serupa dengan yang digunakan dalam kasus zero-sum. Untuk dua pemain, persamaan ini bersifat linear dan dapat diselesaikan dengan teknik pemrograman linear dasar, tetapi untuk tiga pemain atau lebih, persamaan ini bersifat non-linear dan mungkin sangat sulit dipecahkan.

Permainan Berulang

Sejauh ini kita hanya membahas permainan yang berlangsung satu langkah. Jenis permainan multi-langkah yang paling sederhana adalah **permainan berulang**, di mana pemain menghadapi pilihan yang sama berulang kali, tetapi setiap kali dengan pengetahuan tentang

riwayat pilihan semua pemain sebelumnya. Profil strategi untuk permainan berulang menentukan pilihan tindakan untuk setiap pemain pada setiap langkah waktu untuk setiap kemungkinan riwayat pilihan sebelumnya. Seperti halnya MDP, hasil bersifat aditif seiring waktu.

Mari kita pertimbangkan versi berulang dari dilema tahanan. Akankah Alice dan Bob bekerja sama dan menolak untuk bersaksi, meskipun tahu mereka akan bertemu lagi? Jawabannya bergantung pada detail keterlibatan. Misalnya, misalkan Alice dan Bob tahu bahwa mereka harus memainkan tepat 100 putaran dilema tahanan. Maka mereka berdua tahu bahwa putaran ke-100 tidak akan menjadi permainan berulang—artinya, hasilnya tidak akan berpengaruh pada putaran selanjutnya—dan oleh karena itu mereka berdua akan memilih strategi dominan, *bersaksi*, di putaran tersebut. Namun, setelah ronde ke-100 ditentukan, ronde ke-99 tidak akan berpengaruh pada ronde-ronde berikutnya, sehingga ronde tersebut juga akan memiliki keseimbangan strategi dominan pada (*testify, testify*). Dengan induksi, kedua pemain akan memilih *bersaksi* di setiap ronde, yang masing-masing akan mendapatkan hukuman penjara total 500 tahun.

Kita bisa mendapatkan solusi yang berbeda dengan mengubah aturan interaksi. Misalnya, anggaplah setelah setiap putaran ada peluang 99% para pemain akan bertemu lagi. Maka jumlah putaran yang diharapkan tetap 100, tetapi tidak ada pemain yang tahu pasti putaran mana yang akan menjadi yang terakhir. Dalam kondisi ini, perilaku yang lebih kooperatif dimungkinkan. Misalnya, salah satu strategi ekuilibrium adalah setiap pemain *menolak* kecuali pemain lain pernah memainkan *testify*. Strategi ini bisa disebut **hukuman abadi**. Misalkan kedua pemain telah mengadopsi strategi ini, dan ini adalah pengetahuan bersama. Maka selama tidak ada pemain yang memainkan *testify*, maka pada suatu titik waktu, total hasil yang diharapkan di masa depan untuk setiap pemain adalah

$$\sum_{t=0}^{\infty} 0.99^t \cdot (-1) = -100.$$

Pemain yang menyimpang dari strategi dan memilih *bersaksi* akan mendapatkan skor 0 daripada -1 pada langkah berikutnya, tetapi sejak saat itu kedua pemain akan bermain *bersaksi* dan total hasil yang diharapkan di masa depan pemain menjadi

$$0 + \sum_{t=1}^{\infty} 0.99^t \cdot (-5) = -495.$$

Oleh karena itu, di setiap langkah, tidak ada insentif untuk menyimpang dari (*refuse, refuse*). Hukuman abadi adalah strategi "kehancuran bersama yang terjamin" dari dilema tahanan: begitu salah satu pemain memutuskan untuk *bersaksi*, hal itu memastikan kedua pemain sangat menderita. Namun, strategi ini hanya berfungsi sebagai pencegah jika pemain lain

yakin Anda telah mengadopsi strategi ini—atau setidaknya Anda mungkin telah mengadopsinya.

Strategi lain lebih pemaaf. Yang paling terkenal, disebut **tit-for-tat**, mengharuskan dimulai dengan *menolak* dan kemudian mengulangi langkah pemain lain sebelumnya pada semua langkah berikutnya. Jadi, Alice akan menolak selama Bob menolak dan akan bersaksi setelah Bob bersaksi, tetapi akan kembali menolak jika Bob melakukannya. Meskipun sangat sederhana, strategi ini telah terbukti sangat kuat dan efektif melawan berbagai macam strategi.

Kita juga bisa mendapatkan solusi yang berbeda dengan mengubah agen, alih-alih mengubah aturan keterlibatan. Misalkan agen adalah mesin berhingga dengan n keadaan dan mereka memainkan permainan dengan total langkah $m > n$. Para agen dengan demikian tidak mampu merepresentasikan jumlah langkah yang tersisa, dan harus memperlakukannya sebagai sesuatu yang tidak diketahui. Oleh karena itu, mereka tidak dapat melakukan induksi, dan bebas untuk mencapai keseimbangan yang lebih menguntungkan (*refuse, refuse*). Dalam hal ini, ketidaktahuan adalah kebahagiaan—atau lebih tepatnya, membuat lawan Anda percaya bahwa Anda bodoh adalah kebahagiaan. Keberhasilan Anda dalam permainan berulang ini bergantung pada *persepsi* pemain lain tentang Anda sebagai pengganggu atau orang bodoh, dan bukan pada karakteristik Anda yang sebenarnya.

Permainan Berurutan

Pada umumnya, sebuah permainan terdiri dari serangkaian giliran yang tidak harus sama persis. Permainan semacam itu paling baik direpresentasikan oleh pohon permainan, yang oleh para ahli teori permainan disebut **bentuk ekstensif**. Keadaan awal S_0 , fungsi PLAYER(s) yang memberi tahu pemain mana yang memiliki langkah, fungsi ACTIONS(s) yang menghitung kemungkinan tindakan, fungsi RESULT(s, a) yang mendefinisikan transisi ke keadaan baru, dan fungsi parsial UTILITY(s, p), yang didefinisikan hanya pada keadaan akhir, untuk memberikan hasil bagi setiap pemain.

Untuk merepresentasikan permainan stokastik, seperti backgammon, kami menambahkan pemain istimewa, *Chance*, yang dapat melakukan tindakan acak. "Strategi" *Chance* merupakan bagian dari definisi permainan, yang ditentukan sebagai distribusi probabilitas atas tindakan (pemain lain dapat memilih strategi mereka sendiri). Untuk merepresentasikan permainan dengan tindakan nondeterministik, seperti biliard, kami membagi tindakan menjadi dua bagian: tindakan pemain itu sendiri memiliki hasil deterministik, dan kemudian *Chance* memiliki giliran untuk bereaksi terhadap tindakan tersebut dengan caranya sendiri yang berubah-ubah. Untuk merepresentasikan gerakan simultan, seperti dalam dilema tahanan atau Morra dua jari, kami menerapkan urutan acak pada para pemain, tetapi kami memiliki opsi untuk menyatakan bahwa tindakan pemain sebelumnya tidak dapat diamati oleh pemain berikutnya: misalnya, Alice harus memilih *menolak* atau *bersaksi* terlebih dahulu, lalu Bob memilih, tetapi Bob tidak tahu pilihan apa yang diambil Alice saat itu (kami juga dapat merepresentasikan fakta bahwa langkah tersebut terungkap kemudian). Namun, kami berasumsi para pemain selalu mengingat semua tindakan mereka sebelumnya; asumsi ini disebut **ingatan sempurna**.

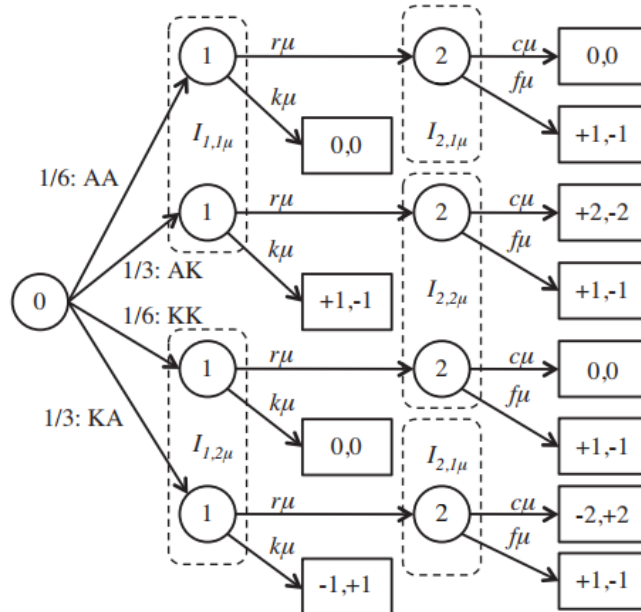
Gagasan utama bentuk ekstensif yang membedakannya dari pohon permainan di Bab 5 adalah representasi observabilitas parsial. Kita telah melihat di Bagian 5.6 bahwa seorang pemain dalam permainan yang dapat diobservasi parsial seperti *Kriegspiel* dapat membuat pohon permainan di atas **ruang status keyakinan**. Dengan pohon tersebut, kita melihat bahwa dalam beberapa kasus seorang pemain dapat menemukan serangkaian langkah (strategi) yang mengarah ke skakmat paksa, terlepas dari status awal kita, dan terlepas dari strategi apa yang digunakan lawan. Namun, teknik-teknik di Bab 5 tidak dapat memberi tahu pemain apa yang harus dilakukan ketika tidak ada skakmat yang dijamin. Jika strategi terbaik pemain bergantung pada strategi lawan dan sebaliknya, maka minimax (atau alfa-beta) sendiri tidak dapat menemukan solusi. Bentuk ekstensif *memungkinkan* kita menemukan solusi karena ia merepresentasikan status keyakinan (para ahli teori permainan menyebutnya **himpunan informasi**) *semua* pemain sekaligus. Dari representasi tersebut, kita dapat menemukan solusi ekuilibrium, sama seperti yang kita lakukan dengan permainan bentuk normal.

Sebagai contoh sederhana permainan sekuensial, tempatkan dua agen di dunia 4×3 pada Gambar 8.1 dan minta mereka bergerak secara simultan hingga salah satu agen mencapai kotak keluar, dan mendapatkan hasil untuk kotak tersebut. Jika kita menetapkan bahwa tidak ada pergerakan yang terjadi ketika kedua agen mencoba bergerak ke kotak yang sama secara bersamaan (masalah umum di banyak persimpangan lalu lintas), maka strategi murni tertentu dapat macet selamanya. Dengan demikian, agen memerlukan strategi campuran agar berkinerja baik dalam permainan ini: memilih secara acak antara bergerak maju dan tetap di tempat. Inilah yang dilakukan untuk mengatasi tabrakan paket dalam jaringan Ethernet.

Selanjutnya, kita akan mempertimbangkan varian poker yang sangat sederhana. Tumpukan kartu hanya terdiri dari empat kartu, dua kartu as dan dua kartu raja. Satu kartu dibagikan kepada setiap pemain. Pemain pertama kemudian memiliki opsi untuk *menaikkan* taruhan permainan dari 1 poin menjadi 2, atau melakukan *skak*. Jika pemain 1 melakukan skak, permainan berakhir. Jika ia *menaikkan*, maka pemain 2 memiliki pilihan untuk *memanggil*, menerima bahwa permainan bernilai 2 poin, atau *melipat*, mengakui 1 poin. Jika permainan tidak berakhir dengan lipatan, maka pembayaran tergantung pada kartu: nol untuk kedua pemain jika mereka memiliki kartu yang sama; jika tidak, pemain dengan raja membayar taruhan kepada pemain dengan as.

Pohon bentuk ekstensif untuk permainan ini ditunjukkan pada Gambar 8.13. Keadaan nonterminal ditunjukkan sebagai lingkaran, dengan pemain untuk bergerak di dalam lingkaran; pemain 0 adalah *peluang*. Setiap tindakan digambarkan sebagai panah dengan label, yang sesuai dengan *kenaikan*, *cek*, *panggilan*, atau *lipatan*, atau, untuk *peluang*, empat kemungkinan pembagian ("AK" berarti bahwa pemain 1 mendapat as dan pemain 2 raja). Keadaan terminal adalah persegi panjang yang diberi label oleh pembayaran mereka kepada pemain 1 dan pemain 2. Kumpulan informasi ditunjukkan sebagai kotak putus-putus berlabel; Misalnya, $I_{1,1}$ adalah himpunan informasi di mana giliran pemain 1, dan ia tahu ia memiliki kartu as (tetapi tidak tahu kartu apa yang dimiliki pemain 2). Dalam himpunan informasi $I_{2,1}$, giliran pemain 2 dan ia tahu ia memiliki kartu as dan pemain 1 telah menaikkan taruhan, tetapi

tidak tahu kartu apa yang dimiliki pemain 1. (Karena keterbatasan kertas dua dimensi, himpunan informasi ini ditampilkan sebagai dua kotak, bukan satu.)



Gambar 8.13 Bentuk luas dari versi poker yang disederhanakan.

Salah satu cara untuk menyelesaikan permainan ekstensif adalah dengan mengubahnya menjadi permainan bentuk normal. Ingat kembali bahwa bentuk normalnya adalah matriks, yang setiap barisnya diberi label strategi murni untuk pemain 1, dan setiap kolom diberi label strategi murni untuk pemain 2. Dalam permainan ekstensif, strategi murni untuk pemain i berkorespondensi dengan tindakan untuk setiap himpunan informasi yang melibatkan pemain tersebut. Jadi, pada Gambar 8.13, salah satu strategi murni untuk pemain 1 adalah "menaikkan taruhan ketika di $I_{1,1}$ (yaitu, ketika saya memiliki kartu as), dan memeriksa ketika di $I_{1,2}$ (ketika saya memiliki kartu raja)." Dalam matriks pembayaran di bawah ini, strategi ini disebut rk . Demikian pula, strategi cf untuk pemain 2 berarti "call ketika saya memiliki kartu as dan fold ketika saya memiliki kartu raja." Karena ini adalah permainan zero-sum, matriks di bawah ini hanya memberikan pembayaran untuk pemain 1; pemain 2 selalu memiliki pembayaran yang berlawanan:

	2:cc	2:cf	2:ff	2:fc
1:rr	0	-1/6	1	7/6
1:kr	-1/3	-1/6	5/6	2/3
1:rk	1/3	0	1/6	1/2
1:kk	0	0	0	0

Permainan ini sangat sederhana sehingga memiliki dua kesetimbangan strategi murni, ditunjukkan dengan huruf tebal: cf untuk pemain 2 dan rk atau kk untuk pemain 1. Namun secara umum, kita dapat menyelesaikan permainan ekstensif dengan mengonversinya ke

bentuk normal, lalu menemukan solusi (biasanya strategi campuran) menggunakan metode pemrograman linier standar. Hal itu berhasil secara teori. Namun, jika seorang pemain memiliki 1 set informasi dan 1 aksi per set, maka pemain tersebut akan memiliki 1 strategi murni. Dengan kata lain, ukuran matriks bentuk normal bersifat eksponensial dalam jumlah set informasi, sehingga dalam praktiknya pendekatan ini hanya berfungsi untuk pohon permainan yang sangat kecil, sekitar selusin negara bagian. Permainan seperti poker Texas Hold'em memiliki sekitar 10^{18} negara bagian, sehingga pendekatan ini sama sekali tidak layak.

Apa saja alternatifnya? Di Bab 5, kita melihat bagaimana pencarian alfa-beta dapat menangani permainan dengan informasi sempurna dan pohon permainan yang besar dengan menghasilkan pohon secara bertahap, memangkas beberapa cabang, dan mengevaluasi simpul nonterminal secara heuristik. Namun, pendekatan tersebut tidak bekerja dengan baik untuk permainan dengan informasi yang tidak sempurna, karena dua alasan: pertama, lebih sulit untuk memangkasnya, karena kita perlu mempertimbangkan strategi campuran yang menggabungkan beberapa cabang, bukan strategi murni yang selalu memilih cabang terbaik. Kedua, lebih sulit untuk mengevaluasi simpul nonterminal secara heuristik, karena kita berurusan dengan kumpulan informasi, bukan status individual.

Koller dan rekan-rekan. (1996) datang untuk menyelamatkan dengan representasi alternatif dari permainan ekstensif, yang disebut **bentuk sekuens**, yang hanya linier dalam ukuran pohon, alih-alih eksponensial. Alih-alih merepresentasikan strategi, bentuk sekuens merepresentasikan jalur melalui pohon; jumlah jalur sama dengan jumlah simpul terminal. Metode pemrograman linier standar dapat diterapkan kembali pada representasi ini. Sistem yang dihasilkan dapat menyelesaikan varian poker dengan 25.000 status dalam satu atau dua menit. Ini merupakan peningkatan kecepatan eksponensial dibandingkan pendekatan bentuk normal, tetapi masih jauh dari menangani poker penuh, dengan 10^{18} status.

Jika kita tidak dapat menangani 10^{18} keadaan, mungkin kita dapat menyederhanakan masalah dengan mengubah permainan ke bentuk yang lebih sederhana. Misalnya, jika saya memegang kartu as dan mempertimbangkan kemungkinan bahwa kartu berikutnya akan menghasilkan sepasang kartu as, maka saya tidak peduli dengan jenis kartu dari kartu berikutnya; jenis kartu apa pun akan sama baiknya. Ini menyarankan pembentukan **abstraksi** permainan, yang mengabaikan jenis kartu. Pohon permainan yang dihasilkan akan lebih kecil dengan faktor $4! = 24$. Misalkan saya dapat menyelesaikan permainan yang lebih kecil ini; bagaimana solusi untuk permainan tersebut akan berhubungan dengan permainan aslinya? Jika tidak ada pemain yang menginginkan flush (atau menggertak), maka jenis kartu tidak menjadi masalah bagi pemain mana pun, dan solusi untuk abstraksi tersebut juga akan menjadi solusi untuk permainan aslinya. Namun, jika ada pemain yang mempertimbangkan flush, maka abstraksi tersebut hanya akan menjadi solusi perkiraan (tetapi batas kesalahan dapat dihitung).

Ada banyak peluang untuk abstraksi. Misalnya, pada titik permainan di mana setiap pemain memiliki dua kartu, jika saya memegang sepasang ratu, maka tangan pemain lain dapat diabstraksikan menjadi tiga kelas: *lebih baik* (hanya sepasang raja atau sepasang as), sama (sepasang ratu) atau *lebih buruk* (semua yang lain). Namun, abstraksi ini mungkin terlalu

kasar. Abstraksi yang lebih baik akan membagi yang lebih *buruk* menjadi, katakanlah, *pasangan sedang* (sembilan hingga jack), *pasangan rendah*, dan *tidak ada pasangan*. Contoh-contoh ini adalah abstraksi dari keadaan; juga memungkinkan untuk mengabstraksikan tindakan. Misalnya, alih-alih memiliki tindakan taruhan untuk setiap bilangan bulat dari 1 hingga 1000, kita dapat membatasi taruhan menjadi 10^0 , 10^1 , 10^2 dan 10^3 . Atau kita dapat memotong salah satu putaran taruhan sama sekali. Kita juga dapat mengabstraksikan simpul peluang lebih, dengan hanya mempertimbangkan sebagian dari kemungkinan pembagian. Ini setara dengan teknik peluncuran yang digunakan dalam program Go. Dengan menggabungkan semua abstraksi ini, kita dapat menyederhanakan 10^{18} keadaan poker menjadi 10^7 keadaan, suatu ukuran yang dapat dipecahkan dengan teknik terkini.

Program poker yang didasarkan pada pendekatan ini dapat dengan mudah mengalahkan pemain pemula dan beberapa pemain manusia berpengalaman, tetapi belum mencapai level pemain master. Sebagian masalahnya adalah solusi yang didekati oleh program ini—solusi ekuilibrium—hanya optimal terhadap lawan yang juga memainkan strategi ekuilibrium. Melawan pemain manusia yang rentan salah, penting untuk dapat mengeksploitasi penyimpangan lawan dari strategi ekuilibrium. Seperti yang dikatakan Gautam Rao (alias "The Count"), pemain poker online terkemuka dunia (Billings dan rekan rekan, 2003), "Anda memiliki program yang sangat kuat. Setelah Anda menambahkan pemodelan lawan ke dalamnya, program itu akan menghancurkan semua orang." Namun, model yang baik tentang kekeliruan manusia masih sulit dipahami.

Dalam arti tertentu, bentuk permainan ekstensif adalah salah satu representasi terlengkap yang pernah kita lihat sejauh ini: ia dapat menangani lingkungan yang dapat diamati sebagian, multiagen, stokastik, sekuensial, dan dinamis—sebagian besar kasus sulit dari daftar properti lingkungan di halaman sebelumnya. Namun, ada dua keterbatasan teori permainan. Pertama, teori ini tidak menangani dengan baik keadaan dan tindakan yang berkelanjutan (meskipun ada beberapa perluasan pada kasus berkelanjutan; misalnya, teori **persaingan Cournot** menggunakan teori permainan untuk memecahkan masalah di mana dua perusahaan memilih harga untuk produk mereka dari ruang berkelanjutan). Kedua, teori permainan mengasumsikan bahwa permainan tersebut *diketahui*. Bagian-bagian permainan mungkin ditentukan sebagai tidak dapat diamati oleh beberapa pemain, tetapi harus diketahui bagian mana yang tidak dapat diamati. Dalam kasus di mana para pemain mempelajari struktur permainan yang tidak diketahui dari waktu ke waktu, model tersebut mulai rusak. Mari kita periksa setiap sumber ketidakpastian, dan apakah masing-masing dapat direpresentasikan dalam teori permainan.

Tindakan: Tidak ada cara mudah untuk merepresentasikan permainan di mana para pemain harus menemukan tindakan apa yang tersedia. Pertimbangkan permainan antara penulis virus komputer dan pakar keamanan. Sebagian dari masalahnya adalah mengantisipasi tindakan apa yang akan dicoba oleh penulis virus selanjutnya.

Strategi: Teori permainan sangat baik dalam merepresentasikan gagasan bahwa strategi pemain lain pada awalnya tidak diketahui—selama kita berasumsi semua agen rasional. Teori itu sendiri tidak mengatakan apa yang harus dilakukan ketika pemain lain

kurang sepenuhnya rasional. Konsep **ekuilibrium Bayes-Nash** sebagian menjawab poin ini: ekuilibrium ini merupakan ekuilibrium yang berkaitan dengan distribusi probabilitas sebelumnya seorang pemain terhadap strategi pemain lain—dengan kata lain, ekuilibrium ini mengekspresikan keyakinan pemain tentang kemungkinan strategi pemain lain.

Peluang: Jika suatu permainan bergantung pada lemparan dadu, cukup mudah untuk memodelkan simpul peluang dengan distribusi seragam atas hasil-hasilnya. Namun, bagaimana jika dadu tersebut mungkin tidak adil? Kita dapat merepresentasikannya dengan simpul peluang lain, yang lebih tinggi di pohon permainan, dengan dua cabang untuk "dadu adil" dan "dadu tidak adil", sehingga simpul-simpul yang bersesuaian di setiap cabang berada dalam himpunan informasi yang sama (artinya, para pemain tidak tahu apakah dadu tersebut adil atau tidak). Dan bagaimana jika kita menduga lawan lain tahu? Kemudian kita menambahkan simpul peluang *lain*, dengan satu cabang mewakili kasus di mana lawan tahu, dan satu cabang lagi di mana ia tidak tahu.

Utilitas: Bagaimana jika kita tidak mengetahui utilitas lawan kita? Sekali lagi, hal itu dapat dimodelkan dengan simpul peluang, sehingga agen lain mengetahui utilitasnya sendiri di setiap cabang, tetapi kita tidak. Namun, bagaimana jika kita tidak mengetahui utilitas kita sendiri? Misalnya, bagaimana saya tahu apakah rasional memesan salad Chef jika saya tidak tahu seberapa besar saya akan menyukainya? Kita dapat memodelkannya dengan simpul peluang lain yang menentukan "kualitas intrinsik" salad yang tidak dapat diamati.

Dengan demikian, kita melihat bahwa teori permainan cukup baik dalam merepresentasikan sebagian besar sumber ketidakpastian—tetapi dengan mengorbankan ukuran pohon yang berlipat ganda setiap kali kita menambahkan simpul lain; sebuah kebiasaan yang dengan cepat menghasilkan pohon yang sangat besar dan sulit diatur. Karena masalah-masalah ini dan masalah lainnya, teori permainan terutama digunakan untuk menganalisis lingkungan yang berada dalam kesetimbangan, alih-alih untuk mengendalikan agen-agen di dalam suatu lingkungan. Selanjutnya, kita akan melihat bagaimana teori permainan dapat membantu merancang lingkungan.

8.6 DESAIN MEKANISME

Pada bagian sebelumnya, kita bertanya, "Jika sebuah permainan, apa yang dimaksud dengan strategi rasional?" Pada bagian ini, kita bertanya, "Jika agen memilih strategi rasional, permainan apa yang harus kita rancang?" Lebih spesifik lagi, kita ingin merancang sebuah permainan yang solusinya, yang terdiri dari setiap agen yang menjalankan strategi rasionalnya sendiri, menghasilkan maksimalisasi beberapa fungsi utilitas global. Masalah ini disebut **desain mekanisme**, atau terkadang **teori permainan terbalik**. Desain mekanisme merupakan pokok ilmu ekonomi dan ilmu politik. Kapitalisme 101 mengatakan bahwa jika setiap orang mencoba menjadi kaya, total kekayaan masyarakat akan meningkat. Namun, contoh-contoh yang akan kita bahas menunjukkan bahwa desain mekanisme yang tepat diperlukan untuk menjaga tangan tak terlihat tetap pada jalurnya. Untuk kumpulan agen, desain mekanisme memungkinkan kita membangun sistem cerdas dari kumpulan sistem yang lebih terbatas—

bahkan sistem yang tidak kooperatif—dengan cara yang hampir sama seperti tim manusia dapat mencapai tujuan di luar jangkauan individu mana pun.

Contoh desain mekanisme meliputi pelelangan tiket pesawat murah, perutean paket TCP antar komputer, penentuan penempatan dokter magang di rumah sakit, dan penentuan bagaimana pemain sepak bola robotik akan bekerja sama dengan rekan satu timnya. Desain mekanisme menjadi lebih dari sekadar subjek akademis pada tahun 1990-an ketika beberapa negara, dihadapkan dengan masalah pelelangan lisensi siaran di berbagai pita frekuensi, kehilangan potensi pendapatan ratusan juta dolar akibat desain mekanisme yang buruk. Secara formal, sebuah mekanisme terdiri dari (1) bahasa untuk mendeskripsikan serangkaian strategi yang diperbolehkan yang dapat diadopsi oleh agen, (2) agen khusus, yang disebut pusat, yang mengumpulkan laporan pilihan strategi dari agen dalam permainan, dan (3) aturan hasil, yang diketahui oleh semua agen, yang digunakan pusat untuk menentukan imbalan bagi setiap agen, berdasarkan pilihan strategi mereka.

Lelang

Mari kita bahas **lelang** terlebih dahulu. Lelang adalah mekanisme untuk menjual sejumlah barang kepada anggota kelompok penawar. Untuk menyederhanakannya, kami berfokus pada lelang dengan satu barang untuk dijual. Setiap penawar i memiliki nilai utilitas v_i untuk memiliki barang tersebut. Dalam beberapa kasus, setiap penawar memiliki **nilai privat** untuk barang tersebut. Misalnya, barang pertama yang dijual di eBay adalah penunjuk laser rusak, yang terjual seharga Rp. 148.300 kepada seorang kolektor penunjuk laser rusak. Dengan demikian, kita tahu bahwa kolektor tersebut memiliki $v_i \geq 148.300$, tetapi kebanyakan orang lain akan memiliki $v_j \ll 148.300$. Dalam kasus lain, seperti melelang hak pengeboran untuk jalur minyak, barang tersebut memiliki **nilai umum**—jalur tersebut akan menghasilkan sejumlah uang, X , dan semua penawar menilai satu dolar secara setara—tetapi terdapat ketidakpastian mengenai nilai X yang sebenarnya. Setiap penawar memiliki informasi yang berbeda, sehingga estimasi nilai sebenarnya dari barang tersebut pun berbeda. Dalam kedua kasus tersebut, penawar akan memiliki v_i mereka sendiri. Dengan v_i , setiap penawar mendapat kesempatan, pada waktu atau waktu-waktu yang tepat dalam lelang, untuk mengajukan penawaran b_i . Penawaran tertinggi, b_{max} , memenangkan barang tersebut, tetapi harga yang dibayarkan tidak harus b_{max} ; itu merupakan bagian dari desain mekanisme.

Mekanisme lelang yang paling terkenal adalah **ascending-bid**, atau **lelang Inggris**, di mana pusat lelang memulai dengan meminta tawaran minimum (atau **cadangan**) b_{min} . Jika ada penawar yang bersedia membayar jumlah tersebut, pusat lelang kemudian meminta $b_{min} + d$, untuk kenaikan tertentu d , dan berlanjut dari sana. Lelang berakhir ketika tidak ada yang mau menawar lagi; kemudian penawar terakhir memenangkan barang tersebut, membayar harga yang ditawarnya.

Bagaimana kita tahu apakah ini mekanisme yang baik? Salah satu tujuannya adalah memaksimalkan pendapatan yang diharapkan bagi penjual. Tujuan lainnya adalah memaksimalkan gagasan tentang utilitas global. Tujuan-tujuan ini saling tumpang tindih sampai batas tertentu, karena salah satu aspek dari memaksimalkan utilitas global adalah memastikan bahwa pemenang lelang adalah agen yang paling menghargai barang tersebut

(dan dengan demikian bersedia membayar paling tinggi). Kita mengatakan lelang **efisien** jika barang tersebut jatuh ke tangan agen yang paling menghargainya. Lelang dengan penawaran menaik biasanya efisien dan memaksimalkan pendapatan, tetapi jika harga cadangan ditetapkan terlalu tinggi, penawar yang paling menghargainya mungkin tidak akan menawar, dan jika harga cadangan ditetapkan terlalu rendah, penjual kehilangan pendapatan bersih.

Mungkin hal terpenting yang dapat dilakukan mekanisme lelang adalah mendorong jumlah penawar yang cukup untuk ikut serta dan mencegah mereka terlibat dalam **kolusi**. Kolusi adalah perjanjian tidak adil atau ilegal oleh dua penawar atau lebih untuk memanipulasi harga. Hal ini dapat terjadi dalam kesepakatan rahasia di balik layar atau secara diam-diam, sesuai aturan mekanisme.

Misalnya, pada tahun 1999, Jerman melelang sepuluh blok spektrum telepon seluler dengan lelang simultan (penawaran dilakukan untuk kesepuluh blok tersebut secara bersamaan), dengan menggunakan aturan bahwa setiap penawaran harus minimal 10% lebih tinggi dari penawaran sebelumnya untuk satu blok. Hanya ada dua penawar yang kredibel, dan yang pertama, Mannesman, memasukkan tawaran sebesar 20 juta mark Jerman pada blok 1-5 dan 18,18 juta pada blok 6-10. Mengapa 18,18 juta? Salah satu manajer T-Mobile mengatakan mereka "menafsirkan tawaran pertama Mannesman sebagai penawaran." Kedua belah pihak dapat menghitung bahwa kenaikan 10% pada 18,18 juta adalah 19,99 juta; dengan demikian tawaran Mannesman ditafsirkan sebagai mengatakan "kita masing-masing bisa mendapatkan setengah blok seharga 20 juta; jangan merusaknya dengan menawar harga lebih tinggi." Dan faktanya T-Mobile menawar 20 juta pada blok 6-10 dan itu adalah akhir dari penawaran. Pemerintah Jerman mendapat kurang dari yang mereka harapkan, karena kedua pesaing dapat menggunakan mekanisme penawaran untuk mencapai kesepakatan diam-diam tentang bagaimana tidak bersaing. Dari sudut pandang pemerintah, hasil yang lebih baik dapat diperoleh dengan salah satu perubahan pada mekanisme ini: harga cadangan yang lebih tinggi; lelang harga pertama dengan penawaran tertutup, sehingga para pesaing tidak dapat berkomunikasi melalui penawaran mereka; atau insentif untuk menghadirkan penawar ketiga. Mungkin aturan 10% merupakan kesalahan dalam desain mekanisme, karena memfasilitasi pengiriman sinyal yang tepat dari Mannesman ke T-Mobile.

Secara umum, baik penjual maupun fungsi utilitas global diuntungkan jika terdapat lebih banyak penawar, meskipun utilitas global dapat menderita jika Anda menghitung biaya waktu yang terbuang dari penawar yang tidak memiliki peluang untuk menang. Salah satu cara untuk mendorong lebih banyak penawar adalah dengan mempermudah mekanismenya bagi mereka. Lagipula, jika memerlukan terlalu banyak riset atau perhitungan dari pihak penawar, mereka mungkin memutuskan untuk mengalihkan uang mereka ke tempat lain. Jadi, sangatlah diinginkan bagi penawar untuk memiliki **strategi yang dominan**. Ingatlah bahwa "dominan" berarti bahwa strategi tersebut bertentangan dengan semua strategi lain, yang pada gilirannya berarti bahwa seorang agen dapat mengadopsinya tanpa memperhatikan strategi lainnya. Seorang agen dengan strategi dominan dapat langsung menawar, tanpa membuang waktu untuk merenungkan kemungkinan strategi agen lain. Mekanisme di mana agen memiliki strategi dominan disebut mekanisme **anti-strategi**. Jika, seperti yang biasanya

terjadi, strategi tersebut melibatkan penawar yang mengungkapkan nilai sebenarnya, v_i , maka itu disebut **lelang yang mengungkapkan kebenaran**, atau **jujur**; istilah yang **kompatibel dengan insentif** juga digunakan. **Prinsip pengungkapan** menyatakan bahwa mekanisme apa pun dapat diubah menjadi mekanisme pengungkapan kebenaran yang ekuivalen, sehingga bagian dari perancangan mekanisme adalah menemukan mekanisme yang ekuivalen ini.

Ternyata lelang dengan penawaran menaik memiliki sebagian besar properti yang diinginkan. Penawar dengan nilai tertinggi v_i mendapatkan barang dengan harga $b_o + d$, di mana b_o adalah penawaran tertinggi di antara semua agen lainnya dan d adalah selisih harga lelang. Para penawar memiliki strategi dominan yang sederhana: terus menawar selama biaya saat ini di bawah v_i Anda. Mekanisme ini tidak sepenuhnya mengungkapkan kebenaran, karena penawar yang menang hanya mengungkapkan bahwa $v_i \geq b_o + d$; kita memiliki batas bawah v_i tetapi bukan jumlah pastinya.

Kerugian (dari sudut pandang penjual) lelang dengan penawaran menaik adalah dapat menghambat persaingan. Misalkan dalam penawaran spektrum telepon seluler, ada satu perusahaan yang diuntungkan yang semua orang sepakat akan mampu memanfaatkan pelanggan dan infrastruktur yang ada, sehingga dapat menghasilkan keuntungan lebih besar daripada perusahaan lain. Calon pesaing dapat melihat bahwa mereka tidak memiliki peluang dalam lelang dengan penawaran menaik, karena perusahaan yang diuntungkan selalu dapat menawar lebih tinggi. Dengan demikian, para pesaing mungkin tidak ikut serta sama sekali, dan perusahaan yang diuntungkan akhirnya menang dengan harga cadangan.

Sifat negatif lain dari lelang Inggris adalah biaya komunikasinya yang tinggi. Lelang dapat berlangsung di satu ruangan atau semua penawar harus memiliki jalur komunikasi berkecepatan tinggi dan aman; dalam kedua kasus tersebut, mereka harus memiliki waktu yang tersedia untuk melalui beberapa putaran penawaran. Mekanisme alternatif, yang membutuhkan komunikasi yang jauh lebih sedikit, adalah **lelang penawaran tertutup**. Setiap penawar mengajukan satu penawaran dan mengomunikasikannya kepada juru lelang, tanpa diketahui oleh penawar lain. Dengan mekanisme ini, tidak ada lagi strategi dominan yang sederhana. Jika nilai Anda adalah v_i dan Anda yakin bahwa tawaran maksimum dari semua agen lain adalah b_o , maka Anda harus menawar $b_o + \epsilon$, untuk beberapa ϵ kecil, jika nilainya kurang dari v_i . Dengan demikian, tawaran Anda bergantung pada estimasi tawaran agen lain, sehingga Anda perlu bekerja lebih keras. Perlu diketahui juga bahwa agen dengan v_i tertinggi mungkin tidak memenangkan lelang. Hal ini diimbangi oleh fakta bahwa lelang lebih kompetitif, sehingga mengurangi bias terhadap penawar yang diuntungkan.

Perubahan kecil dalam mekanisme lelang tertutup menghasilkan **lelang harga kedua tertutup**, yang juga dikenal sebagai lelang Vickrey.¹⁰ Dalam lelang semacam itu, pemenang membayar harga penawaran tertinggi *kedua*, b_o , alih-alih membayar penawarannya sendiri. Modifikasi sederhana ini sepenuhnya menghilangkan pertimbangan rumit yang diperlukan untuk lelang tertutup standar (atau **harga pertama**), karena strategi yang dominan sekarang hanyalah menawar v_i ; mekanismenya mengungkapkan kebenaran. Perhatikan bahwa utilitas agen i dalam hal penawarannya b_i , nilainya v_i , dan penawaran terbaik di antara agen lainnya, b_o , adalah

$$u_i = \begin{cases} (v_i - b_o) & \text{if } b_i > b_o \\ 0 & \text{otherwise.} \end{cases}$$

Untuk melihat bahwa $b_i = v_i$ merupakan strategi dominan, perhatikan bahwa ketika $(v_i - b_o)$ positif, setiap tawaran yang memenangkan lelang adalah optimal, dan khususnya tawaran v_i memenangkan lelang. Di sisi lain, ketika $(v_i - b_o)$ negatif, setiap tawaran yang kalah dalam lelang adalah optimal, dan khususnya tawaran v_i kalah dalam lelang. Jadi, tawaran v_i optimal untuk semua kemungkinan nilai b_o , dan faktanya, v_i adalah satu-satunya tawaran yang memiliki sifat ini. Karena kesederhanaannya dan persyaratan komputasi minimal bagi penjual dan penawar, lelang Vickrey banyak digunakan dalam membangun sistem AI terdistribusi. Selain itu, mesin pencari internet melakukan lebih dari satu miliar lelang setiap hari untuk menjual iklan beserta hasil pencariannya, dan situs lelang daring menangani barang senilai \$100 miliar per tahun, semuanya menggunakan varian lelang Vickrey. Perhatikan bahwa nilai yang diharapkan bagi penjual adalah b_o , yang merupakan imbal hasil yang diharapkan sama dengan batas lelang Inggris ketika selisih d menjadi nol. Ini sebenarnya hasil yang sangat umum: **Teorema Ekuivalensi Pendapatan** menyatakan bahwa, dengan beberapa peringatan kecil, mekanisme lelang apa pun di mana penawar yang netral terhadap risiko memiliki nilai v_i yang hanya diketahui oleh mereka sendiri (tetapi mengetahui distribusi probabilitas dari mana nilai-nilai tersebut diambil sampelnya), akan menghasilkan pendapatan yang diharapkan sama. Prinsip ini berarti bahwa berbagai mekanisme tidak bersaing berdasarkan perolehan pendapatan, melainkan pada kualitas lain.

Meskipun lelang harga kedua mengungkapkan kebenaran, ternyata memperluas gagasan tersebut ke beberapa barang dan menggunakan lelang harga berikutnya bukanlah pengungkapan kebenaran. Banyak mesin pencari internet menggunakan mekanisme di mana mereka melelang k slot untuk iklan di satu halaman. Penawar tertinggi memenangkan posisi teratas, penawar tertinggi kedua mendapatkan posisi kedua, dan seterusnya. Setiap pemenang membayar harga yang ditawarkan oleh penawar terendah berikutnya, dengan pemahaman bahwa pembayaran hanya dilakukan jika pencari benar-benar mengklik iklan tersebut. Slot teratas dianggap lebih berharga karena lebih mungkin diperhatikan dan diklik. Bayangkan tiga penawar, b_1, b_2 , dan b_3 , memiliki valuasi untuk klik $v_1 = 200$, $v_2 = 180$, dan $v_3 = 100$, dan $k = 2$ slot tersedia, di mana diketahui bahwa tempat teratas diklik 5% dari waktu dan tempat terbawah 2%. Jika semua penawar menawar dengan jujur, maka b_1 memenangkan slot teratas dan membayar 180, dan memiliki pengembalian yang diharapkan sebesar $(200 - 180) \times 0,05 = 1$. Slot kedua jatuh ke b_2 . Namun, b_1 dapat melihat bahwa jika ia menawar apa pun dalam rentang 101–179, ia akan menyerahkan slot teratas kepada b_2 , memenangkan slot kedua, dan menghasilkan pengembalian yang diharapkan sebesar $(200 - 100) \times 0,02 = 2$. Jadi, b_1 dapat menggandakan pengembalian yang diharapkan dengan menawar kurang dari nilai sebenarnya dalam kasus ini. Secara umum, penawar dalam lelang multislot ini harus menghabiskan banyak energi menganalisis tawaran orang lain untuk menentukan strategi terbaik mereka; tidak ada strategi dominan yang sederhana. Aggarwal

dan rekan rekan. (2006) menunjukkan bahwa terdapat mekanisme lelang jujur yang unik untuk masalah multislot ini, di mana pemenang slot j membayar harga penuh untuk slot j hanya untuk klik tambahan yang tersedia di slot j dan bukan di slot $j + 1$. Pemenang membayar harga untuk slot bawah untuk klik yang tersisa. Dalam contoh kami, b_1 akan menawar 200 dengan jujur, dan akan membayar 180 untuk tambahan $0,05 - 0,02 = 0,03$ klik di slot teratas, tetapi hanya akan membayar biaya slot bawah, 100, untuk sisa 0,02 klik. Dengan demikian, total pengembalian untuk b_1 adalah $(200 - 180) \times 0,03 + (200 - 100) \times 0,02 = 2,6$.

Contoh lain di mana lelang dapat berperan dalam AI adalah ketika sekelompok agen memutuskan apakah akan bekerja sama dalam rencana bersama. Hunsberger dan Grosz (2000) menunjukkan bahwa hal ini dapat dicapai secara efisien dengan lelang di mana para agen menawar peran dalam rencana bersama.

Barang Umum

Sekarang mari kita pertimbangkan jenis permainan lain, di mana negara-negara menetapkan kebijakan mereka untuk mengendalikan polusi udara. Setiap negara memiliki pilihan: mereka dapat mengurangi polusi dengan biaya -10 poin untuk menerapkan perubahan yang diperlukan, atau mereka dapat terus mencemari, yang memberi mereka utilitas bersih -5 (dalam bentuk biaya kesehatan tambahan, dll.) dan juga berkontribusi -1 poin untuk setiap negara lain (karena udara digunakan bersama oleh berbagai negara). Jelas, strategi dominan untuk setiap negara adalah "terus mencemari", tetapi jika ada 100 negara dan masing-masing mengikuti kebijakan ini, maka setiap negara mendapatkan utilitas total -104, sedangkan jika setiap negara mengurangi polusi, masing-masing akan memiliki utilitas -10. Situasi ini disebut **tragedi milik bersama**: jika tidak ada yang harus membayar untuk menggunakan sumber daya umum, maka sumber daya tersebut cenderung dieksploitasi dengan cara yang menghasilkan utilitas total yang lebih rendah bagi semua pihak. Hal ini serupa dengan dilema tahanan: ada solusi lain untuk permainan ini yang lebih baik bagi semua pihak, tetapi tampaknya tidak ada cara bagi agen rasional untuk mencapai solusi tersebut.

Pendekatan standar untuk menangani tragedi sumber daya bersama adalah dengan mengubah mekanisme menjadi mekanisme yang membebankan biaya kepada setiap agen atas penggunaan sumber daya bersama. Secara lebih umum, kita perlu memastikan bahwa semua **eksternalitas**—dampak pada utilitas global yang tidak dikenali dalam transaksi masing-masing agen—dijelaskan. Menetapkan harga dengan tepat adalah bagian yang sulit. Pada dasarnya, pendekatan ini sama dengan menciptakan mekanisme di mana setiap agen secara efektif diharuskan untuk memaksimalkan utilitas global, tetapi dapat melakukannya dengan membuat keputusan lokal. Untuk contoh ini, pajak karbon akan menjadi contoh mekanisme yang membebankan biaya atas penggunaan sumber daya bersama dengan cara yang, jika diterapkan dengan baik, akan memaksimalkan utilitas global.

Sebagai contoh terakhir, pertimbangkan masalah alokasi beberapa barang bersama. Misalkan sebuah kota memutuskan ingin memasang beberapa transceiver internet nirkabel gratis. Namun, jumlah transceiver yang mereka mampu beli lebih sedikit daripada jumlah lingkungan yang menginginkannya. Pemerintah kota ingin mengalokasikan barang secara efisien, ke lingkungan yang paling menghargainya. Artinya, mereka ingin memaksimalkan

utilitas global $V = \sum_i v_i$. Masalahnya, jika mereka hanya bertanya kepada setiap dewan lingkungan, "Berapa nilai hadiah gratis ini?", mereka semua akan terdorong untuk berbohong dan melaporkan nilai yang tinggi. Ternyata ada mekanisme, yang dikenal sebagai mekanisme **Vickrey-Clarke-Groves**, atau **VCG**, yang menjadikannya strategi dominan bagi setiap agen untuk melaporkan utilitas sebenarnya dan mencapai alokasi barang yang efisien. Triknya adalah setiap agen membayar pajak yang setara dengan kerugian utilitas global yang terjadi karena kehadiran agen dalam permainan. Mekanismenya bekerja seperti ini:

1. Pusat meminta setiap agen untuk melaporkan nilainya karena menerima suatu barang. Sebut ini b_i .
2. Pusat mengalokasikan barang kepada sebagian penawar. Kami menyebut subset ini A , dan menggunakan notasi $b_i(A)$ untuk mengartikan hasil ke i dalam alokasi ini: b_i jika i ada di A (artinya, i adalah pemenang), dan 0 jika tidak. Pusat memilih A untuk memaksimalkan total utilitas yang dilaporkan $B = \sum_i b_i(A)$.
3. Pusat menghitung (untuk setiap i) jumlah utilitas yang dilaporkan untuk semua pemenang kecuali i . Kami menggunakan notasi $B_{-i} = \sum_{j \neq i} b_j(A)$. Pusat juga menghitung (untuk setiap i) alokasi yang akan memaksimalkan total utilitas global jika i tidak ada dalam permainan; sebut jumlah itu W_{-i} .
4. Setiap agen i membayar pajak sebesar $W_{-i} - B_{-i}$.

Dalam contoh ini, aturan VCG berarti setiap pemenang akan membayar pajak yang setara dengan nilai tertinggi yang dilaporkan di antara para pecundang. Artinya, jika saya melaporkan nilai saya sebagai 5, dan hal itu menyebabkan seseorang dengan nilai 2 kehilangan alokasi, maka saya membayar pajak sebesar 2. Semua pemenang seharusnya senang karena mereka membayar pajak yang lebih rendah dari nilai mereka, dan semua pecundang sebahagia mungkin, karena mereka menilai barang tersebut kurang dari pajak yang diwajibkan.

Mengapa mekanisme ini mengungkapkan kebenaran? Pertama, pertimbangkan imbalan bagi agen i , yaitu nilai mendapatkan suatu barang, dikurangi pajak:

$$v_i(A) - (W_{-i} - B_{-i}). \quad (8.14)$$

Di sini kita membedakan utilitas sejati agen, v_i , dari utilitas yang dilaporkannya, b_i (tetapi kita mencoba menunjukkan bahwa strategi yang dominan adalah $b_i = v_i$). Agen i tahu bahwa pusat akan memaksimalkan utilitas global menggunakan nilai-nilai yang dilaporkan,

$$\sum_j b_j(A) = b_i(A) + \sum_{j \neq i} b_j(A)$$

sedangkan agen i ingin pusat memaksimalkan (17.14), yang dapat ditulis ulang menjadi

$$v_i(A) + \sum_{j \neq i} b_j(A) - W_{-i}.$$

Karena agen i tidak dapat memengaruhi nilai W_{-i} (hanya bergantung pada agen lain), satu-satunya cara agar pusat mengoptimalkan apa yang diinginkan i adalah dengan melaporkan utilitas sebenarnya, $b_i = v_i$.

8.7 RINGKASAN

Bab ini menunjukkan cara menggunakan pengetahuan tentang dunia untuk membuat keputusan, bahkan ketika hasil suatu tindakan tidak pasti dan imbalan atas tindakan tersebut mungkin baru dirasakan setelah banyak tindakan berlalu. Poin-poin utamanya adalah sebagai berikut:

- Masalah keputusan sekuensial dalam lingkungan yang tidak pasti, juga disebut proses **keputusan Markov**, atau MDP, didefinisikan oleh **model transisi** yang menentukan hasil probabilistik dari tindakan dan **fungsi imbalan** yang menentukan imbalan di setiap keadaan.
- Utilitas suatu urutan keadaan adalah jumlah semua imbalan selama urutan tersebut, yang mungkin didiskonkan seiring waktu. Solusi MDP adalah **kebijakan** yang mengaitkan suatu keputusan dengan setiap keadaan yang mungkin dicapai agen. Kebijakan optimal memaksimalkan utilitas urutan keadaan yang ditemui ketika kebijakan tersebut dieksekusi.
- Utilitas suatu keadaan adalah utilitas yang diharapkan dari urutan keadaan yang ditemui ketika suatu kebijakan optimal dieksekusi, dimulai dari keadaan tersebut. Algoritma **iterasi nilai** untuk menyelesaikan MDP bekerja dengan memecahkan persamaan secara iteratif yang menghubungkan utilitas setiap keadaan dengan keadaan tetangganya.
- **Iterasi kebijakan** bergantian antara menghitung utilitas keadaan di bawah kebijakan saat ini dan meningkatkan kebijakan saat ini sehubungan dengan utilitas saat ini.
- MDP yang dapat diobservasi sebagian, atau POMDP, jauh lebih sulit dipecahkan daripada MDP. MDP dapat dipecahkan dengan konversi ke MDP dalam ruang kontinu keadaan keyakinan; baik algoritma iterasi nilai maupun iterasi kebijakan telah dirancang. Perilaku optimal dalam POMDP mencakup pengumpulan informasi untuk mengurangi ketidakpastian dan dengan demikian membuat keputusan yang lebih baik di masa mendatang.
- Agen teoretis-keputusan dapat dibangun untuk lingkungan POMDP. Agen menggunakan **jaringan keputusan dinamis** untuk merepresentasikan model transisi dan sensor, untuk memperbarui keadaan keyakinannya, dan untuk memproyeksikan kemungkinan urutan tindakan ke depan.
- **Teori permainan** menggambarkan perilaku rasional agen dalam situasi di mana beberapa agen berinteraksi secara bersamaan. Solusi permainan adalah **ekuilibria Nash**—profil strategi di mana tidak ada agen yang memiliki insentif untuk menyimpang dari strategi yang ditentukan.
- **Desain mekanisme** dapat digunakan untuk menetapkan aturan interaksi antar agen, guna memaksimalkan utilitas global melalui pengoperasian masing-masing agen yang

rasional. Terkadang, terdapat mekanisme yang mencapai tujuan ini tanpa mengharuskan setiap agen mempertimbangkan pilihan yang dibuat oleh agen lain.

BAB 9

BELAJAR DARI CONTOH

Seorang agen dikatakan belajar jika ia meningkatkan kinerjanya pada tugas-tugas mendatang setelah melakukan pengamatan terhadap dunia. Pembelajaran dapat berkisar dari hal sepele, seperti yang ditunjukkan dengan mencatat nomor telepon, hingga hal mendalam, seperti yang ditunjukkan oleh Albert Einstein, yang menyimpulkan teori baru tentang alam semesta. Dalam bab ini, kita akan berfokus pada satu jenis masalah pembelajaran, yang tampaknya terbatas tetapi sebenarnya memiliki penerapan yang luas: dari sekumpulan pasangan masukan-keluaran, mempelajari fungsi yang memprediksi keluaran untuk masukan baru. Mengapa kita ingin agen belajar? Jika desain agen dapat ditingkatkan, mengapa perancang tidak langsung memprogram peningkatan tersebut sejak awal? Ada tiga alasan utama. Pertama, perancang tidak dapat mengantisipasi semua kemungkinan situasi yang mungkin dihadapi agen. Misalnya, robot yang dirancang untuk menavigasi labirin harus mempelajari tata letak setiap labirin baru yang ditemuinya. Kedua, perancang tidak dapat mengantisipasi semua perubahan dari waktu ke waktu; program yang dirancang untuk memprediksi harga saham besok harus belajar beradaptasi ketika kondisi berubah dari naik turun. Ketiga, terkadang pemrogram manusia tidak tahu cara memprogram solusi sendiri. Misalnya, kebanyakan orang pandai mengenali wajah anggota keluarga, tetapi bahkan programmer terbaik pun tidak mampu memprogram komputer untuk menyelesaikan tugas tersebut, kecuali dengan menggunakan algoritma pembelajaran. Bab ini pertama-tama memberikan gambaran umum tentang berbagai bentuk pembelajaran, kemudian menjelaskan satu pendekatan populer, pembelajaran pohon keputusan, di Bagian 9.3, diikuti dengan analisis teoretis pembelajaran di Bagian 9.4 dan 9.5. Kami mengkaji berbagai sistem pembelajaran yang digunakan dalam praktik: model linear, model nonlinier (khususnya, jaringan saraf tiruan), model nonparametrik, dan mesin vektor pendukung. Terakhir, kami menunjukkan bagaimana ansambel model dapat mengungguli satu model.

9.1 BENTUK PEMBELAJARAN

Setiap komponen agen dapat ditingkatkan dengan belajar dari data. Peningkatan tersebut, dan teknik yang digunakan untuk membuatnya, bergantung pada empat faktor utama:

- *Komponen* mana yang akan ditingkatkan.
- *Pengetahuan awal* apa yang telah dimiliki agen.
- *Representasi* apa yang digunakan untuk data dan komponen.
- *Umpan balik* apa yang tersedia untuk dipelajari.

Komponen yang akan dipelajari

Bab 2 menjelaskan beberapa desain agen. Komponen-komponen agen ini meliputi:

1. Pemetaan langsung dari kondisi pada keadaan saat ini ke tindakan.

2. Sarana untuk menyimpulkan properti dunia yang relevan dari urutan persepsi.
3. Informasi tentang bagaimana dunia berevolusi dan tentang hasil dari kemungkinan tindakan yang dapat diambil agen.
4. Informasi utilitas yang menunjukkan keinginan keadaan dunia.
5. Informasi nilai-tindakan yang menunjukkan keinginan tindakan.
6. Sasaran yang menggambarkan kelas keadaan yang pencapaiannya memaksimalkan utilitas agen.

Masing-masing komponen ini dapat dipelajari. Pertimbangkan, misalnya, seorang agen yang berlatih untuk menjadi sopir taksi. Setiap kali instruktur berteriak "Rem!", agen mungkin mempelajari aturan kondisi-tindakan untuk kapan harus mengerem (komponen 1); Agen juga belajar setiap kali instruktur tidak berteriak. Dengan melihat banyak gambar kamera yang diberi tahu berisi bus, agen dapat belajar mengenalinya (2). Dengan mencoba tindakan dan mengamati hasilnya—misalnya, mengerem mendadak di jalan basah—agen dapat mempelajari efek dari tindakannya (3). Kemudian, ketika tidak menerima informasi dari penumpang yang terguncang hebat selama perjalanan, agen dapat mempelajari komponen yang berguna dari keseluruhan fungsi utilitasnya (4).

Representasi dan Pengetahuan Awal

Kita telah melihat beberapa contoh representasi untuk komponen agen: kalimat logis proposisional dan orde pertama untuk komponen-komponen dalam agen logis; jaringan Bayesian untuk komponen inferensial agen teoretis-keputusan, dan seterusnya. Algoritma pembelajaran yang efektif telah dirancang untuk semua representasi ini. Bab ini (dan sebagian besar penelitian pembelajaran mesin saat ini) membahas masukan yang membentuk **representasi terfaktor**—sebuah vektor nilai atribut—dan keluaran yang dapat berupa nilai numerik kontinu atau nilai diskrit.

Ada cara lain untuk melihat berbagai jenis pembelajaran. Kita mengatakan bahwa mempelajari fungsi atau aturan umum (yang mungkin salah) dari pasangan masukan-keluaran tertentu disebut **pembelajaran induktif**. Kita akan melihat di Bab ini bahwa kita juga dapat melakukan **pembelajaran analitis** atau **deduktif**: beralih dari aturan umum yang diketahui ke aturan baru yang secara logis disiratkan, tetapi bermanfaat karena memungkinkan pemrosesan yang lebih efisien.

Umpan balik untuk dipelajari

Ada tiga jenis umpan balik yang menentukan tiga jenis utama pembelajaran:

Dalam **pembelajaran tanpa pengawasan**, agen mempelajari pola dalam masukan meskipun tidak ada umpan balik eksplisit yang diberikan. Tugas pembelajaran tanpa pengawasan yang paling umum adalah **pengelompokan**: mendeteksi klaster contoh masukan yang berpotensi bermanfaat. Misalnya, seorang agen taksi mungkin secara bertahap mengembangkan konsep "hari lalu lintas baik" dan "hari lalu lintas buruk" tanpa pernah diberi contoh berlabel untuk masing-masing oleh guru.

Dalam **pembelajaran penguatan**, agen belajar dari serangkaian penguatan—hadiah atau hukuman. Misalnya, tidak adanya tip di akhir perjalanan memberi agen taksi indikasi bahwa ia melakukan kesalahan. Dua poin untuk kemenangan di akhir permainan catur

memberi tahu agen bahwa ia melakukan sesuatu yang benar. Terserah agen untuk memutuskan tindakan mana sebelum penguatan yang paling bertanggung jawab atas kesalahan tersebut.

Dalam **pembelajaran terbimbing**, agen mengamati beberapa contoh pasangan masukan-keluaran dan mempelajari fungsi yang memetakan dari masukan ke keluaran. Pada komponen 1 di atas, masukannya adalah persepsi dan keluarannya diberikan oleh guru yang berkata, "Rem!" atau "Belok kiri." Pada komponen 2, masukannya adalah gambar kamera dan keluarannya lagi-lagi berasal dari guru yang berkata, "Itu bus." Pada komponen 3, teori pengereman merupakan fungsi dari keadaan dan aksi pengereman terhadap jarak berhenti dalam kaki. Dalam hal ini, nilai keluaran tersedia langsung dari persepsi agen (setelah kejadian); lingkungannya adalah guru.

Dalam praktiknya, perbedaan ini tidak selalu begitu jelas. Dalam **pembelajaran semi-supervised**, kita diberikan beberapa contoh berlabel dan harus memanfaatkan sebisa mungkin kumpulan besar contoh yang tidak berlabel. Bahkan label itu sendiri mungkin bukan kebenaran orakular yang kita harapkan. Bayangkan Anda sedang mencoba membangun sistem untuk menebak usia seseorang dari sebuah foto. Anda mengumpulkan beberapa contoh berlabel dengan memotret orang-orang dan menanyakan usia mereka. Itulah pembelajaran tersupervised. Namun kenyataannya, beberapa orang berbohong tentang usia mereka. Bukan hanya karena terdapat gangguan acak dalam data; melainkan ketidakakuratan tersebut bersifat sistematis, dan mengungkapkannya merupakan masalah pembelajaran tanpa pengawasan yang melibatkan gambar, usia yang dilaporkan sendiri, dan usia sebenarnya (yang tidak diketahui). Dengan demikian, gangguan dan ketiadaan label menciptakan kontinum antara pembelajaran terawasi dan tanpa pengawasan.

9.2 PEMBELAJARAN TERAWASI

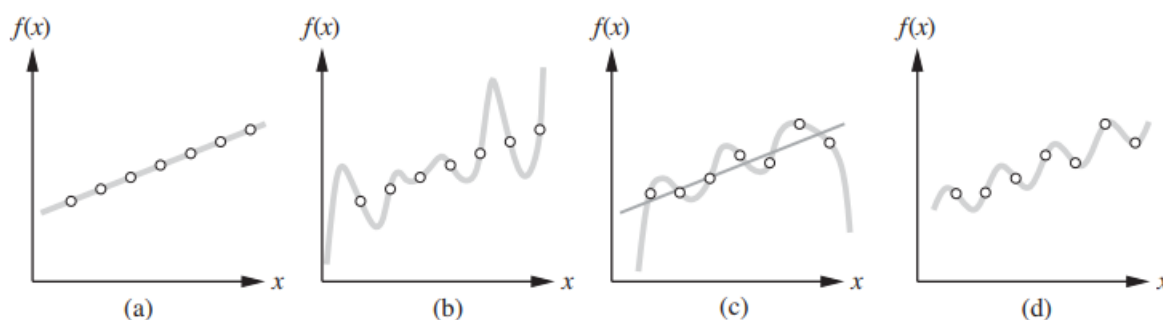
Tugas pembelajaran terawasi adalah sebagai berikut:

Diberikan satu **set pelatihan** yang terdiri dari N contoh pasangan input-output

$$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N),$$

di mana setiap y_j dihasilkan oleh fungsi $y = f(x)$ yang tidak diketahui, temukan fungsi h yang mengaproksimasi fungsi f yang sebenarnya.

Di sini, x dan y dapat berupa nilai apa pun; keduanya tidak harus berupa angka. Fungsi h adalah sebuah **hipotesis**. Pembelajaran adalah pencarian melalui ruang hipotesis yang memungkinkan untuk menemukan hipotesis yang akan berkinerja baik, bahkan pada contoh-contoh baru di luar set pelatihan. Untuk mengukur akurasi sebuah hipotesis, kami memberinya **set uji** contoh-contoh yang berbeda dari set pelatihan. Kami mengatakan sebuah hipotesis **tergeneralisasi** dengan baik jika ia memprediksi nilai y dengan tepat untuk contoh-contoh baru. Terkadang fungsi f bersifat stokastik—bukanlah fungsi x secara ketat, dan yang harus kita pelajari adalah distribusi probabilitas bersyarat, $\mathbf{P}(Y | x)$.



Gambar 9.1 (a) Contoh pasangan $(x, f(x))$ dan hipotesis linier yang konsisten. (b) Hipotesis polinomial derajat 7 yang konsisten untuk kumpulan data yang sama. (c) Kumpulan data yang berbeda, yang memungkinkan kecocokan polinomial derajat 6 yang tepat atau kecocokan linier perkiraan. (d) Kecocokan sinusoidal yang tepat dan sederhana untuk kumpulan data yang sama.

Ketika keluaran y adalah salah satu dari sekumpulan nilai yang terbatas (seperti *sunny, cloudy or rainy*), masalah pembelajaran disebut **klasifikasi**, dan disebut Boolean atau klasifikasi biner jika hanya ada dua nilai. Ketika y adalah angka (seperti suhu besok), masalah pembelajaran disebut **regresi**. (Secara teknis, menyelesaikan masalah regresi berarti menemukan ekspektasi bersyarat atau nilai rata-rata y , karena probabilitas kita menemukan angka riil yang tepat untuk y adalah 0.).

Gambar 9.1 menunjukkan contoh yang umum: pencocokan fungsi variabel tunggal ke beberapa titik data. Contohnya adalah titik-titik pada bidang (x, y) , di mana $y = f(x)$. Kita tidak tahu apa itu f , tetapi kita akan mengaproksimasinya dengan fungsi h yang dipilih dari **ruang hipotesis**, $\mathcal{H}$, yang dalam contoh ini kita anggap sebagai himpunan polinomial, seperti $x^5 + 3x^2 + 2$. Gambar 9.1(a) menunjukkan beberapa data dengan kecocokan eksak melalui garis lurus (polinomial $0,4x + 3$). Garis tersebut disebut hipotesis **konsisten** karena sesuai dengan semua data. Gambar 9.1(b) menunjukkan polinomial derajat tinggi yang juga konsisten dengan data yang sama. Hal ini menggambarkan masalah mendasar dalam pembelajaran induktif: *bagaimana kita memilih dari beberapa hipotesis yang konsisten?* Salah satu jawabannya adalah memilih hipotesis paling sederhana yang konsisten dengan data. Prinsip ini disebut pisau **cukur Ockham**, diambil dari nama filsuf Inggris abad ke-14, William Ockham, yang menggunakannya untuk berargumen tajam menentang segala macam kerumitan. Mendefinisikan kesederhanaan tidaklah mudah, tetapi tampak jelas bahwa polinomial derajat 1 lebih sederhana daripada polinomial derajat 7, sehingga (a) seharusnya lebih diutamakan daripada (b). Kita akan membuat intuisi ini lebih tepat di Bagian selanjutnya

Gambar 9.1(c) menunjukkan set data kedua. Tidak ada garis lurus yang konsisten untuk set data ini; bahkan, dibutuhkan polinomial derajat 6 untuk kecocokan yang tepat. Hanya ada 7 titik data, sehingga polinomial dengan 7 parameter tampaknya tidak menemukan pola apa pun dalam data dan kami tidak berharap polinomial tersebut dapat digeneralisasi dengan baik. Garis lurus yang tidak konsisten dengan titik data mana pun, tetapi mungkin dapat digeneralisasi dengan cukup baik untuk nilai x yang tidak terlihat, juga ditunjukkan pada (c). Secara umum, terdapat *trade-off* antara hipotesis kompleks yang sesuai dengan data

pelatihan dengan baik dan hipotesis yang lebih sederhana yang mungkin dapat digeneralisasi dengan lebih baik. Pada Gambar 9.1(d), kami memperluas ruang hipotesis $\mathcal{H}$ untuk memungkinkan polinomial pada x dan $\sin(x)$, dan menemukan bahwa data pada (c) dapat difitting secara tepat oleh fungsi sederhana berbentuk $ax + b + c\sin(x)$. Hal ini menunjukkan pentingnya pemilihan ruang hipotesis. Kami mengatakan bahwa masalah pembelajaran dapat **direalisasikan** jika ruang hipotesis memuat fungsi yang benar. Sayangnya, kita tidak selalu dapat memastikan apakah suatu masalah pembelajaran dapat direalisasikan, karena fungsi sebenarnya tidak diketahui.

Dalam beberapa kasus, seorang analis yang mengamati suatu masalah bersedia membuat perbedaan yang lebih rinci tentang ruang hipotesis, untuk mengatakan—bahkan sebelum melihat data apa pun—bukan hanya apakah suatu hipotesis mungkin atau tidak mungkin, tetapi juga seberapa besar kemungkinannya. Pembelajaran terbimbing dapat dilakukan dengan memilih hipotesis h^* yang paling mungkin berdasarkan data:

$$h^* = \operatorname{argmax}_{h \in \mathcal{H}} P(h|data).$$

Menurut aturan Bayes, hal ini setara dengan

$$h^* = \operatorname{argmax}_{h \in \mathcal{H}} P(data|h)P(h).$$

Maka kita dapat mengatakan bahwa probabilitas prior $P(h)$ tinggi untuk polinomial derajat 1 atau -2, lebih rendah untuk polinomial derajat 7, dan khususnya rendah untuk polinomial derajat 7 dengan lonjakan besar dan tajam seperti pada Gambar 9.1(b). Kita mengizinkan fungsi yang tampak tidak biasa ketika data menunjukkan bahwa kita benar-benar membutuhkannya, tetapi kita mencegahnya dengan memberinya probabilitas prior yang rendah.

Mengapa tidak menjadikan $\mathcal{H}$ sebagai kelas dari semua program Java, atau mesin Turing? Lagipula, setiap fungsi komputasi dapat direpresentasikan oleh beberapa mesin Turing, dan itulah yang terbaik yang dapat kita lakukan. Salah satu masalah dengan gagasan ini adalah gagasan ini tidak memperhitungkan kompleksitas komputasional pembelajaran. *Ada trade-off antara ekspresivitas ruang hipotesis dan kompleksitas menemukan hipotesis yang baik dalam ruang tersebut.* Misalnya, memasang garis lurus pada data merupakan komputasi yang mudah; memasang polinomial derajat tinggi agak lebih sulit; dan memasang mesin Turing pada umumnya tidak dapat diputuskan. Alasan kedua untuk lebih memilih ruang hipotesis sederhana adalah karena kita mungkin ingin menggunakan h setelah mempelajarinya, dan menghitung $h(x)$ ketika h adalah fungsi linear dijamin cepat, sementara menghitung program mesin Turing arbitrer bahkan tidak dijamin selesai. Karena alasan ini, sebagian besar penelitian tentang pembelajaran berfokus pada representasi sederhana.

Kita akan melihat bahwa tradeoff antara ekspresivitas dan kompleksitas tidak sesederhana yang terlihat: sering kali, seperti yang kita lihat dengan logika orde pertama di

Bab 8, bahasa ekspresif memungkinkan hipotesis *sederhana* untuk menyesuaikan dengan data, sementara membatasi ekspresivitas bahasa berarti bahwa setiap hipotesis yang konsisten harus sangat kompleks. Misalnya, aturan catur dapat ditulis dalam satu atau dua halaman logika orde pertama, tetapi membutuhkan ribuan halaman ketika ditulis dalam logika proposisional.

9.3 POHON KEPUTUSAN PEMBELAJARAN

Induksi pohon keputusan adalah salah satu bentuk pembelajaran mesin yang paling sederhana namun paling berhasil. Pertama-tama, kami menjelaskan representasinya—ruang hipotesis—dan kemudian menunjukkan cara mempelajari hipotesis yang baik.

Representasi Pohon Keputusan

Pohon keputusan merepresentasikan suatu fungsi yang menerima vektor nilai atribut sebagai input dan mengembalikan sebuah "keputusan"—sebuah nilai output tunggal. Nilai input dan output dapat berupa diskrit atau kontinu. Untuk saat ini, kita akan berfokus pada permasalahan di mana input memiliki nilai diskrit dan output memiliki tepat dua kemungkinan nilai; ini adalah klasifikasi Boolean, di mana setiap contoh input akan diklasifikasikan sebagai benar (contoh **positif**) atau salah (contoh **negatif**).

Sebuah pohon keputusan mencapai keputusannya dengan melakukan serangkaian pengujian. Setiap simpul internal di dalam pohon tersebut bersesuaian dengan pengujian nilai salah satu atribut input, A_i , dan cabang-cabang dari simpul tersebut diberi label dengan kemungkinan nilai atribut tersebut, $A_i = v_{ik}$. Setiap simpul daun di dalam pohon tersebut menentukan nilai yang akan dikembalikan oleh fungsi. Representasi pohon keputusan bersifat alami bagi manusia; bahkan, banyak buku panduan "Cara Melakukan" (misalnya, untuk perbaikan mobil) ditulis sepenuhnya sebagai satu pohon keputusan tunggal yang membentang hingga ratusan halaman.

Sebagai contoh, kita akan membangun pohon keputusan untuk memutuskan apakah akan menunggu meja di restoran. Tujuannya adalah mempelajari definisi **predikat tujuan** *WillWait*. Pertama, kita akan membuat daftar atribut yang akan kita pertimbangkan sebagai bagian dari input:

1. *Alternatif*: apakah ada restoran alternatif yang cocok di dekatnya.
2. *Bar*: apakah restoran memiliki area bar yang nyaman untuk menunggu.
3. *Fri/Sat*: benar pada hari Jumat dan Sabtu.
4. *Hungry*: apakah kita lapar.
5. *Patrons*: berapa banyak orang di restoran (nilainya adalah *Tidak Ada*, *Beberapa*, dan *Penuh*).
6. *Price*: kisaran harga restoran (\$, \$\$, \$\$\$).
7. *Raining*: apakah hujan di luar.
8. *Reservation*: apakah kita telah melakukan reservasi.
9. *Type*: jenis restoran (Prancis, Italia, Thailand, atau burger).
10. *WaitEstimate*: perkiraan waktu tunggu oleh host (0–10 menit, 10–30, 30–60, atau >60).

Perhatikan bahwa setiap variabel memiliki sekumpulan kecil kemungkinan nilai; nilai *WaitEstimate*, misalnya, bukanlah bilangan bulat, melainkan salah satu dari empat nilai diskrit 0–10, 10–30, 30–60, atau >60. Pohon keputusan yang biasanya digunakan oleh salah satu dari kami (SR) untuk domain ini ditunjukkan pada Gambar 9.2. Perhatikan bahwa pohon keputusan tersebut mengabaikan atribut *Harga* dan *Jenis*. Contoh diproses oleh pohon keputusan mulai dari akar dan mengikuti cabang yang sesuai hingga mencapai daun. Misalnya, contoh dengan *Patrons = Full* dan *WaitEstimate = 0 – 10* akan diklasifikasikan sebagai positif (yaitu, ya, kami akan menunggu meja).

Ekspresivitas Pohon Keputusan

Pohon keputusan Boolean secara logis setara dengan pernyataan bahwa atribut tujuan bernilai benar jika dan hanya jika atribut input memenuhi salah satu jalur menuju daun bernilai *benar*. Jika dituliskan dalam logika proposisional, kita memiliki

$$Goal \Leftrightarrow (Path_1 \vee Path_2 \vee \dots),$$

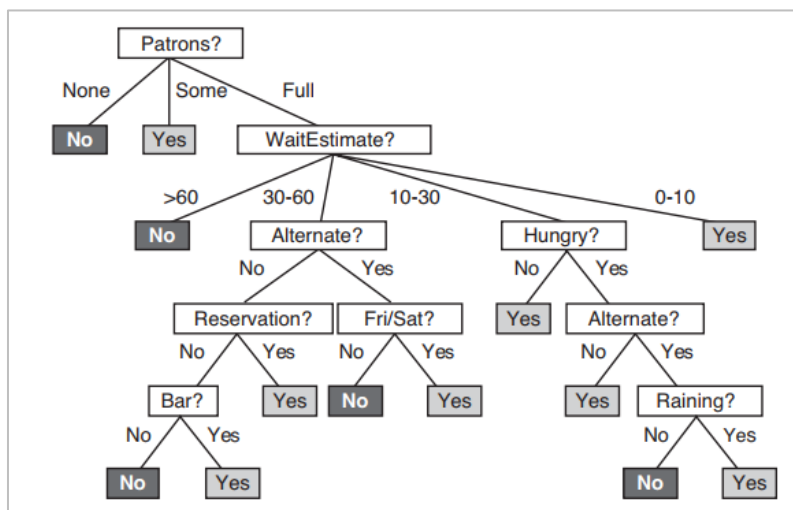
Keseluruhan ekspresi ini setara dengan bentuk normal disjungtif, yang berarti bahwa fungsi apa pun dalam logika proposisional dapat diekspresikan sebagai pohon keputusan. Sebagai contoh, jalur paling kanan pada Gambar 9.2 adalah

$$Path = (Patrons = Full \wedge WaitEstimate = 0 - 10).$$

Untuk berbagai macam masalah, format pohon keputusan menghasilkan hasil yang bagus dan ringkas. Namun, beberapa fungsi tidak dapat direpresentasikan secara ringkas. Misalnya, fungsi mayoritas, yang mengembalikan nilai true jika dan hanya jika lebih dari separuh input bernilai true, membutuhkan pohon keputusan yang besarnya eksponensial. Dengan kata lain, pohon keputusan baik untuk beberapa jenis fungsi dan buruk untuk fungsi lainnya. Adakah representasi yang efisien untuk *semua* jenis fungsi? Sayangnya, jawabannya tidak. Kita dapat menunjukkan hal ini secara umum. Perhatikan himpunan semua fungsi Boolean pada n atribut. Berapa banyak fungsi berbeda dalam himpunan ini? Ini hanyalah jumlah tabel kebenaran berbeda yang dapat kita tulis, karena fungsi didefinisikan oleh tabel kebenarannya. Sebuah tabel kebenaran dengan n atribut memiliki 2^n baris, satu untuk setiap kombinasi nilai atribut. Kita dapat menganggap kolom "jawaban" dari tabel tersebut sebagai angka 2^n -bit yang mendefinisikan fungsi tersebut. Artinya, terdapat 2^n fungsi berbeda (dan akan ada lebih dari jumlah pohon tersebut, karena lebih dari satu pohon dapat menghitung fungsi yang sama). Ini angka yang menakutkan. Misalnya, hanya dengan sepuluh atribut Boolean dari permasalahan restoran kita, terdapat 2^{1024} atau sekitar 10^{308} fungsi berbeda yang dapat dipilih, dan untuk 20 atribut terdapat lebih dari $10^{300.000}$. Kita membutuhkan beberapa algoritma yang cerdas untuk menemukan hipotesis yang baik dalam ruang sebesar itu.

Menginduksi pohon keputusan dari contoh

Sebuah contoh untuk pohon keputusan Boolean terdiri dari pasangan (x, y) , dengan x adalah vektor nilai untuk atribut masukan, dan y adalah nilai keluaran Boolean tunggal. Set pelatihan yang terdiri dari 12 contoh ditunjukkan pada Gambar 9.3. Contoh positif adalah contoh di mana tujuan *WillWait* bernilai benar ($x_1, x_3, \dots$); contoh negatif adalah contoh di mana tujuannya salah ($x_2, x_5, \dots$).



Gambar 9.2 Pohon keputusan untuk memutuskan apakah akan menunggu meja.

Tabel 18.3 Contoh untuk domain restoran.

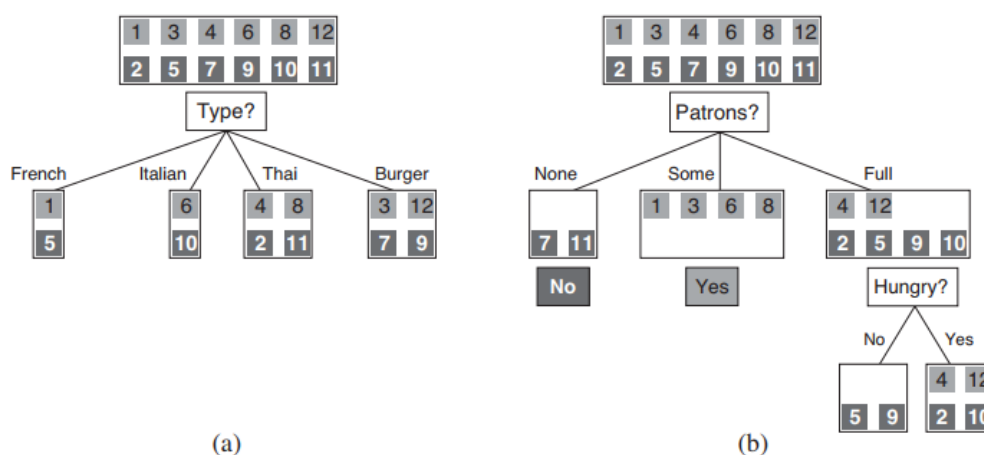
Contoh	Atribut Masukan										Tujuan
	Alt	Bar	frid	Hun	Pat	Harga	Hujan	Res	Tipe	Est	
x1	Ya	Tidak	Tidak	Ya	Beberapa	\$\$\$	Tidak	Ya	Prancis	0-10	x1
x2	Ya	Tidak	Tidak	Tidak	Penuh	\$	Tidak	Tidak	Thai	30-60	x2
x3	Tidak	Ya	Tidak	Ya	Beberapa	\$	Tidak	Tidak	Burger	0-10	x3
x4	Ya	Tidak	Ya	Ya	Penuh	\$	Ya	Tidak	Thai	10-30	x4
x5	Ya	Tidak	Ya	Tidak	Penuh	\$\$\$	Tidak	Ya	Prancis	>60	x5
x6	Tidak	Ya	Ya	Ya	Beberapa	\$\$	Ya	Ya	Italia	0-10	x6
x7	Tidak	Ya	Tidak	Tidak	Tidak ada	\$	Ya	Tidak	Burger	0-10	x7
x8	Tidak	Ya	Tidak	Ya	Beberapa	\$\$	Ya	Tidak	Thai	0-10	x8
x9	Tidak	Tidak	Ya	Tidak	Penuh	\$	Ya	Ya	Burger	>60	x9
x10	Ya	Ya	Ya	Ya	Penuh	\$\$\$	Tidak	Ya	Italia	10-30	x10
x11	Tidak	Tidak	Tidak	Tidak	Tidak ada	\$	Tidak	Tidak	Thai	0-10	x11
x12	Ya	Ya	Ya	Ya	Penuh	\$	Tidak	Tidak	Burger	30-60	x12

Kita menginginkan pohon yang konsisten dengan contoh dan sekecil mungkin. Sayangnya, tidak peduli bagaimana kita mengukur ukuran, itu adalah masalah yang sulit diatasi untuk menemukan pohon konsisten terkecil; tidak ada cara untuk mencari secara efisien melalui 2^{2^n} pohon. Namun, dengan beberapa heuristik sederhana, kita dapat menemukan solusi perkiraan yang baik: pohon konsisten kecil (tetapi bukan yang terkecil). Algoritma PEMBELAJARAN POHON KEPUTUSAN mengadopsi strategi membagi-dan-menaklukkan yang rakus: selalu menguji atribut yang paling penting terlebih dahulu. Pengujian ini membagi masalah menjadi submasalah yang lebih kecil yang kemudian dapat dipecahkan secara

rekursif. Dengan "atribut yang paling penting," yang kami maksud adalah yang membuat perbedaan paling besar pada klasifikasi suatu contoh. Dengan begitu, kami berharap untuk mendapatkan klasifikasi yang benar dengan sejumlah kecil pengujian, yang berarti bahwa semua jalur di pohon akan pendek dan pohon secara keseluruhan akan dangkal.

Gambar 9.4(a) menunjukkan bahwa *Type* adalah atribut yang buruk, karena menyisakan empat kemungkinan hasil, yang masing-masing memiliki jumlah contoh positif dan negatif yang sama. Di sisi lain, pada (b) kita melihat bahwa *Patrons* adalah atribut yang cukup penting, karena jika nilainya *None* atau *Some*, maka kita akan memiliki kumpulan contoh yang dapat kita jawab dengan pasti (*No* dan *Yes*, berturut-turut). Jika nilainya *Full*, kita akan memiliki kumpulan contoh yang beragam. Secara umum, setelah uji atribut pertama membagi contoh-contoh tersebut, setiap hasil merupakan masalah pembelajaran pohon keputusan baru, dengan lebih sedikit contoh dan satu atribut lebih sedikit. Ada empat kasus yang perlu dipertimbangkan untuk masalah rekursif ini:

1. Jika semua contoh yang tersisa positif (atau semua negatif), maka kita selesai: kita dapat menjawab *Yes* atau *No*. Gambar 9.4(b) menunjukkan contoh-contoh yang terjadi di cabang *None* dan *Some*.
2. Jika ada beberapa contoh positif dan beberapa contoh negatif, maka pilih atribut terbaik untuk membaginya. Gambar 9.4(b) menunjukkan *Hungry* digunakan untuk memisahkan contoh-contoh yang tersisa.
3. Jika tidak ada contoh yang tersisa, artinya tidak ada contoh yang diamati untuk kombinasi nilai atribut ini, dan kami mengembalikan nilai default yang dihitung dari klasifikasi pluralitas semua contoh yang digunakan dalam membangun induk simpul. Nilai-nilai ini diteruskan dalam variabel *parent_examples*.
4. Jika tidak ada atribut yang tersisa, tetapi terdapat contoh positif dan negatif, artinya contoh-contoh ini memiliki deskripsi yang persis sama, tetapi klasifikasinya berbeda. Hal ini dapat terjadi karena terdapat kesalahan atau **gangguan** dalam data; karena domainnya nondeterministik; atau karena kami tidak dapat mengamati atribut yang dapat membedakan contoh-contoh tersebut. Hal terbaik yang dapat kami lakukan adalah mengembalikan klasifikasi pluralitas dari contoh-contoh yang tersisa.



Gambar 9.4 Memisahkan contoh dengan pengujian atribut. Pada setiap simpul, kami menampilkan contoh positif (kotak terang) dan negatif (kotak gelap) yang tersisa. (a) Memisahkan berdasarkan *Tipe* tidak membantu kita membedakan antara contoh positif dan negatif. (b) Memisahkan berdasarkan *Patron* berhasil memisahkan contoh positif dan negatif dengan baik. Setelah membagi berdasarkan *Patron*, *Hungry* merupakan pengujian kedua yang cukup baik.

Algoritma PEMBELAJARAN POHON KEPUTUSAN ditunjukkan pada Gambar 9.5. Perhatikan bahwa kumpulan contoh sangat penting untuk *membangun* pohon, tetapi contoh-contoh tersebut tidak muncul di pohon itu sendiri. Sebuah pohon hanya terdiri dari pengujian atribut di simpul interior, nilai atribut di cabang, dan nilai keluaran di simpul daun. Detail fungsi IMPORTANCE diberikan di Bagian 9.3.4. Keluaran algoritma pembelajaran pada set pelatihan contoh kami ditunjukkan pada Gambar 9.6. Pohon ini jelas berbeda dari pohon asli yang ditunjukkan pada Gambar 9.2. Kita mungkin menyimpulkan bahwa algoritma pembelajaran tidak bekerja dengan baik dalam mempelajari fungsi yang benar. Namun, ini adalah kesimpulan yang salah. Algoritma pembelajaran melihat *contoh*, bukan fungsi yang benar, dan faktanya, hipotesisnya (lihat Gambar 9.6) tidak hanya konsisten dengan semua contoh, tetapi juga jauh lebih sederhana daripada pohon asli! Algoritma pembelajaran tidak memiliki alasan untuk menyertakan pengujian untuk *Raining* dan *Reservation*, karena ia dapat mengklasifikasikan semua contoh tanpa keduanya. Algoritma ini juga mendeteksi pola yang menarik dan sebelumnya tidak terduga: penulis pertama akan menunggu makanan Thailand di akhir pekan. Algoritma ini juga pasti akan membuat beberapa kesalahan untuk kasus-kasus di mana ia tidak melihat contoh. Misalnya, belum pernah ditemukan kasus di mana waktu tungguanya 0–10 menit tetapi restorannya penuh.

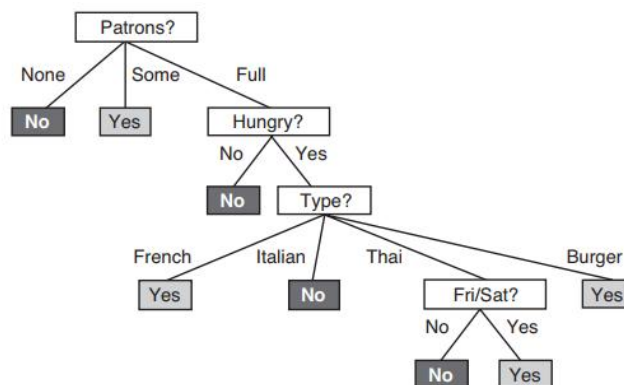
```

function DECISION – TREE – LEARNING(examples, attributes, parent examples) returns a tree

  if examples is empty then return PLURALITY – VALUE(parent examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY – VALUE(examples)
  else
     $A \leftarrow \operatorname{argmax}_{a \in \text{attributes}} \text{IMPORTANCE}(a, \text{examples})$ 
    tree  $\leftarrow$  a new decision tree with root test A
    for each value  $v_k$  of A do
      exs  $\leftarrow \{e : e \in \text{examples} \text{ and } e.A = v_k\}$ 
      subtree  $\leftarrow$  DECISION – TREE – LEARNING(exs, attributes – A, examples)
      add a branch to tree with label (A =  $v_k$ ) and subtree subtree
    return tree

```

Gambar 9.5 Algoritma pembelajaran pohon keputusan. Fungsi IMPORTANCE dijelaskan di Bagian 9.3.. Fungsi PLURALITY-VALUE memilih nilai keluaran yang paling umum di antara sekumpulan contoh, dan memutuskan nilai seri secara acak.

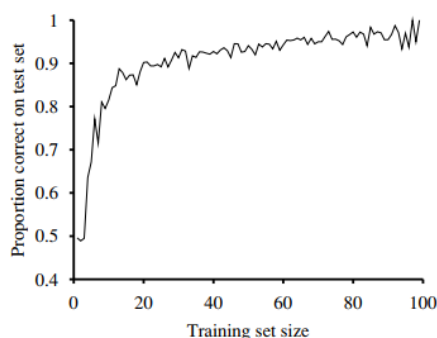


Gambar 9.6 Pohon keputusan yang diinduksi dari set pelatihan 12 contoh.

Dalam kasus tersebut, algoritma tersebut menyatakan untuk tidak menunggu ketika *Hungry* bernilai salah, tetapi saya (SR) tentu saja akan menunggu. Dengan lebih banyak contoh pelatihan, program pembelajaran dapat memperbaiki kesalahan ini.

Kami mencatat adanya bahaya penafsiran berlebihan terhadap pohon yang dipilih algoritma. Ketika terdapat beberapa variabel dengan kepentingan yang sama, pilihan di antara variabel-variabel tersebut agak arbitrer: dengan contoh masukan yang sedikit berbeda, variabel yang berbeda akan dipilih untuk dipecah terlebih dahulu, dan keseluruhan pohon akan terlihat sangat berbeda. Fungsi yang dihitung oleh pohon akan tetap serupa, tetapi struktur pohonnya dapat sangat bervariasi.

Kita dapat mengevaluasi akurasi algoritma pembelajaran dengan **kurva pembelajaran**, seperti yang ditunjukkan pada Gambar 9.7. Kita memiliki 100 contoh yang tersedia, yang kita bagi menjadi satu set pelatihan dan satu set pengujian. Kita mempelajari hipotesis h dengan set pelatihan dan mengukur akurasinya dengan set pengujian. Kami melakukan ini dengan memulai set pelatihan berukuran 1 dan meningkatkannya satu per satu hingga mencapai ukuran 99. Untuk setiap ukuran, kami mengulangi proses pemisahan acak sebanyak 20 kali, dan menghitung rata-rata hasil dari 20 percobaan. Kurva menunjukkan bahwa seiring bertambahnya ukuran set pelatihan, akurasinya pun meningkat. (Oleh karena itu, kurva pembelajaran juga disebut **grafik bahagia**.) Dalam grafik ini, akurasi kami mencapai 95%, dan tampaknya kurva tersebut akan terus meningkat seiring bertambahnya data.



Gambar 9.7 Kurva pembelajaran untuk algoritma pembelajaran pohon keputusan pada 100 contoh yang dihasilkan secara acak di domain restoran. Setiap titik data merupakan rata-rata dari 20 percobaan.

Memilih uji atribut

Pencarian greedy yang digunakan dalam pembelajaran pohon keputusan dirancang untuk meminimalkan kedalaman pohon akhir. Idennya adalah memilih atribut yang sedapat mungkin memberikan klasifikasi contoh yang tepat. Atribut sempurna membagi contoh menjadi himpunan, yang masing-masing positif atau negatif, sehingga akan menjadi daun pohon. Atribut *Patrons* tidaklah sempurna, tetapi cukup baik. Atribut yang benar-benar tidak berguna, seperti *Type*, meninggalkan himpunan contoh dengan proporsi contoh positif dan negatif yang kurang lebih sama dengan himpunan aslinya.

Jadi, yang kita butuhkan hanyalah ukuran formal "cukup baik" dan "benar-benar tidak berguna" dan kita dapat mengimplementasikan fungsi IMPORTANCE pada Gambar 9.5. Kita akan menggunakan gagasan perolehan informasi, yang didefinisikan dalam istilah **entropi**, kuantitas fundamental dalam teori informasi.

Entropi adalah ukuran ketidakpastian variabel acak; perolehan informasi bersesuaian dengan pengurangan entropi. Variabel acak dengan hanya satu nilai—koin yang selalu muncul gambar—tidak memiliki ketidakpastian sehingga entropinya didefinisikan sebagai nol; oleh karena itu, kita tidak memperoleh informasi apa pun dengan mengamati nilainya. Lemparan koin yang adil memiliki kemungkinan yang sama untuk muncul gambar atau ekor, 0 atau 1, dan kita akan segera menunjukkan bahwa ini dihitung sebagai "1 bit" entropi. Lemparan dadu *empat* sisi yang adil memiliki 2 bit entropi, karena dibutuhkan dua bit untuk menggambarkan satu dari empat pilihan yang sama kemungkinannya. Sekarang perhatikan koin yang tidak adil yang muncul gambar 99% dari waktu. Secara intuitif, koin ini memiliki ketidakpastian yang lebih rendah daripada koin yang adil—jika kita menebak gambar, kita hanya akan salah 1% dari waktu—jadi kita ingin koin tersebut memiliki ukuran entropi yang mendekati nol, tetapi positif. Secara umum, entropi variabel acak V dengan nilai v_k , masing-masing dengan probabilitas $P(v_k)$, didefinisikan sebagai

$$\text{Entropy: } H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = - \sum_k P(v_k) \log_2 P(v_k).$$

Kita dapat memeriksa bahwa entropi lemparan koin yang adil memang 1 bit:

$$H(\text{Fair}) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1.$$

Jika koin diisi untuk memberikan 99% kepala, kita mendapatkan

$$H(\text{Loaded}) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits}.$$

Ini akan membantu untuk mendefinisikan $B(q)$ sebagai entropi variabel acak Boolean yang benar dengan probabilitas q :

$$B(q) = -(q \log_2 q + (1 - q) \log_2 (1 - q)).$$

Dengan demikian, $H(\text{Loaded}) = B(0.99) \approx 0.08$. Sekarang mari kita kembali ke pembelajaran pohon keputusan. Jika suatu set pelatihan berisi p contoh positif dan n contoh negatif, maka entropi atribut tujuan pada keseluruhan set adalah

$$H(\text{Goal}) = B\left(\frac{p}{p+n}\right).$$

Set pelatihan restoran pada Gambar 9.3 memiliki $p = n = 6$, sehingga entropi yang sesuai adalah $B(0.5)$ atau tepat 1 bit. Pengujian pada satu atribut A mungkin hanya memberi kita sebagian dari 1 bit ini. Kita dapat mengukur seberapa banyak dengan melihat entropi yang tersisa setelah pengujian atribut.

Atribut A dengan d nilai berbeda membagi set pelatihan E menjadi subset $E_1, \dots, E_d$. Setiap subset E_k memiliki p_k contoh positif dan n_k contoh negatif, jadi jika kita mengikuti cabang itu, kita akan membutuhkan tambahan $B(p_k/(p_k + n_k))$ bit informasi untuk menjawab pertanyaan. Contoh yang dipilih secara acak dari set pelatihan memiliki nilai ke- k untuk atribut dengan probabilitas $(p_k + n_k)/(p + n)$, sehingga entropi yang diharapkan tersisa setelah pengujian atribut A adalah

$$\text{Remainder}(A) = \sum_{k=1}^d \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right).$$

Keuntungan informasi dari uji atribut pada A adalah pengurangan entropi yang diharapkan.:

$$\text{Gain}(A) = B\left(\frac{p}{p+n}\right) - \text{Remainder}(A).$$

Faktanya, $\text{Gain}(A)$ adalah apa yang kita butuhkan untuk mengimplementasikan fungsi IMPORTANCE. Kembali ke atribut yang dipertimbangkan pada Gambar 9.4, kita memiliki

$$\text{Gain}(\text{Patrons}) = 1 - \left[\frac{2}{12} B\left(\frac{0}{2}\right) + \frac{4}{12} B\left(\frac{4}{4}\right) + \frac{6}{12} B\left(\frac{2}{6}\right) \right] \approx 0.541 \text{ bits},$$

$$\text{Gain}(\text{Type}) = 1 - \left[\frac{2}{12} B\left(\frac{1}{2}\right) + \frac{2}{12} B\left(\frac{1}{2}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) + \frac{4}{12} B\left(\frac{2}{4}\right) \right] = 0 \text{ bits},$$

Ini mengonfirmasi intuisi kami bahwa *Patron* adalah atribut yang lebih baik untuk dipecah. Faktanya, *Patron* memiliki perolehan maksimum dibandingkan atribut lainnya dan akan dipilih oleh algoritma pembelajaran pohon keputusan sebagai akarnya.

Generalisasi dan overfitting

Pada beberapa permasalahan, algoritma PEMBELAJARAN POHON KEPUTUSAN akan menghasilkan pohon besar padahal sebenarnya tidak ada pola yang ditemukan. Pertimbangkan permasalahan mencoba memprediksi apakah lemparan dadu akan menghasilkan angka 6 atau tidak. Misalkan eksperimen dilakukan dengan berbagai dadu dan atribut yang menggambarkan setiap contoh pelatihan meliputi warna dadu, beratnya, waktu lemparan, dan apakah para peneliti menebak dengan benar. Jika dadunya adil, hal yang tepat untuk dipelajari adalah pohon dengan satu simpul yang mengatakan "tidak". Namun, algoritma PEMBELAJARAN POHON KEPUTUSAN akan memanfaatkan pola apa pun yang ditemukannya dalam input. Jika ternyata terdapat 2 lemparan dadu biru seberat 7 gram dengan jari disilangkan dan keduanya menghasilkan angka 6, maka algoritma dapat membangun jalur yang memprediksi angka 6 dalam kasus tersebut. Permasalahan ini disebut **overfitting**. Overfitting, sebuah fenomena umum, terjadi pada semua jenis pembelajar, bahkan ketika fungsi target sama sekali tidak acak. Pada Gambar 9.1(b) dan (c), kita melihat fungsi polinomial melakukan overfitting pada data. Overfitting menjadi lebih mungkin terjadi seiring bertambahnya ruang hipotesis dan jumlah atribut input, dan semakin kecil kemungkinannya seiring bertambahnya jumlah contoh pelatihan.

Untuk pohon keputusan, teknik yang disebut **pemangkasan pohon keputusan** mengatasi overfitting. Pemangkasan bekerja dengan mengeliminasi simpul yang tidak relevan secara jelas. Kita mulai dengan pohon penuh, seperti yang dihasilkan oleh PEMBELAJARAN POHON KEPUTUSAN. Kemudian kita melihat simpul uji yang hanya memiliki simpul daun sebagai turunannya. Jika pengujian tersebut tampak tidak relevan—hanya mendeteksi derau dalam data—maka kita mengeliminasi pengujian tersebut, menggantinya dengan simpul daun. Kita mengulangi proses ini, dengan mempertimbangkan setiap pengujian yang hanya memiliki turunan daun, hingga setiap simpul telah dipangkas atau diterima apa adanya.

Pertanyaannya adalah, bagaimana kita mendeteksi bahwa sebuah simpul sedang menguji atribut yang tidak relevan? Misalkan kita berada pada sebuah simpul yang terdiri dari p contoh positif dan n contoh negatif. Jika atribut tersebut tidak relevan, kita akan memperkirakan bahwa atribut tersebut akan membagi contoh-contoh menjadi subset yang masing-masing memiliki proporsi contoh positif yang kurang lebih sama dengan keseluruhan set, $p/(p + n)$, sehingga perolehan informasi akan mendekati nol.² Dengan demikian, perolehan informasi merupakan petunjuk yang baik untuk ketidakrelevanan. Sekarang pertanyaannya adalah, seberapa besar perolehan yang kita perlukan untuk membagi atribut tertentu?

Kita dapat menjawab pertanyaan ini dengan menggunakan **uji signifikansi statistik**. Uji tersebut dimulai dengan asumsi bahwa tidak ada pola yang mendasarinya (yang disebut **hipotesis nol**). Kemudian, data aktual dianalisis untuk menghitung sejauh mana data tersebut menyimpang dari ketiadaan pola yang sempurna. Jika tingkat penyimpangan secara statistik

tidak mungkin (biasanya dianggap berarti probabilitas 5% atau kurang), maka hal itu dianggap sebagai bukti yang baik untuk keberadaan pola yang signifikan dalam data. Probabilitas dihitung dari distribusi standar dari jumlah penyimpangan yang diharapkan terlihat dalam pengambilan sampel acak.

Dalam kasus ini, hipotesis nolnya adalah bahwa atribut tersebut tidak relevan dan, oleh karena itu, perolehan informasi untuk sampel yang sangat besar akan menjadi nol. Kita perlu menghitung probabilitas bahwa, berdasarkan hipotesis nol, sampel berukuran $v = n + p$ akan menunjukkan deviasi yang teramati dari distribusi ekspektasi contoh positif dan negatif. Kita dapat mengukur deviasi tersebut dengan membandingkan jumlah aktual contoh positif dan negatif di setiap subset, p_k dan n_k , dengan jumlah ekspektasi, $\widehat{p}_k$ dan $\widehat{n}_k$, dengan asumsi ketidakrelevanan sejati:

$$\widehat{p}_k = p \times \frac{p_k + n_k}{p + n} \quad \widehat{n}_k = n \times \frac{p_k + n_k}{p + n}.$$

Ukuran yang mudah untuk mengetahui deviasi total diberikan oleh

$$\Delta = \sum_{k=1}^d \left(\frac{(p_k - \widehat{p}_k)^2}{\widehat{p}_k} + \frac{(n_k - \widehat{n}_k)^2}{\widehat{n}_k} \right).$$

Berdasarkan hipotesis nol, nilai Δ terdistribusi menurut distribusi χ^2 (chi-kuadrat) dengan derajat kebebasan $v - 1$. Kita dapat menggunakan tabel χ^2 atau rutin pustaka statistik standar untuk melihat apakah nilai Δ tertentu mengonfirmasi atau menolak hipotesis nol. Misalnya, perhatikan atribut jenis restoran, dengan empat nilai dan dengan demikian tiga derajat kebebasan. Nilai $\Delta = 7,82$ atau lebih akan menolak hipotesis nol pada tingkat 5% (dan nilai $\Delta = 11,35$ atau lebih akan menolak pada tingkat 1%).

Dengan pemangkasan, gangguan dalam contoh-contoh tersebut dapat ditoleransi. Kesalahan pada label contoh (misalnya, contoh $(\mathbf{x}, \text{Yes})$ yang seharusnya $(\mathbf{x}, \text{No})$) menyebabkan peningkatan linear dalam kesalahan prediksi, sedangkan kesalahan dalam deskripsi contoh (misalnya, $\text{Harga} = \$$ padahal sebenarnya $\text{Harga} = \$\$$) memiliki efek asimtotik yang semakin parah seiring menyusutnya pohon menjadi himpunan yang lebih kecil. Pohon yang dipangkas berkinerja jauh lebih baik daripada pohon yang tidak dipangkas ketika data mengandung banyak gangguan. Selain itu, pohon yang dipangkas seringkali jauh lebih kecil sehingga lebih mudah dipahami.

Satu peringatan terakhir: Anda mungkin berpikir bahwa pemangkasan χ^2 dan perolehan informasi terlihat serupa, jadi mengapa tidak menggabungkannya menggunakan pendekatan yang disebut **penghentian dini**—membiarkan algoritma pohon keputusan berhenti menghasilkan simpul ketika tidak ada atribut yang baik untuk dipecah, daripada bersusah payah menghasilkan simpul lalu memangkasnya. Masalah dengan penghentian dini adalah ia menghalangi kita mengenali situasi di mana tidak ada satu atribut yang baik, tetapi

terdapat kombinasi atribut yang informatif. Misalnya, perhatikan fungsi XOR dari dua atribut biner. Jika terdapat jumlah contoh yang kurang lebih sama untuk keempat kombinasi nilai input, maka tidak ada atribut yang akan informatif. Namun, hal yang benar untuk dilakukan adalah membagi pada salah satu atribut (apa pun atributnya), dan kemudian pada tingkat kedua kita akan mendapatkan pembagian yang informatif. Penghentian awal akan melewatkan hal ini, tetapi pembangkitan-lalu-pemangkasan menanganinya dengan tepat.

Memperluas Penerapan Pohon Keputusan

Untuk memperluas induksi pohon keputusan ke berbagai masalah yang lebih luas, sejumlah masalah harus diatasi. Kami akan membahas beberapa secara singkat, menyarankan bahwa pemahaman yang lengkap paling baik diperoleh dengan melakukan latihan terkait:

- **Data yang hilang:** Dalam banyak domain, tidak semua nilai atribut akan diketahui untuk setiap contoh. Nilai-nilai tersebut mungkin tidak tercatat, atau mungkin terlalu mahal untuk diperoleh. Hal ini menimbulkan dua masalah: Pertama, dengan pohon keputusan yang lengkap, bagaimana seseorang harus mengklasifikasikan contoh yang kehilangan salah satu atribut uji? Kedua, bagaimana seseorang harus memodifikasi rumus perolehan informasi ketika beberapa contoh memiliki nilai yang tidak diketahui untuk atribut tersebut?
- **Atribut bernilai banyak:** Ketika suatu atribut memiliki banyak kemungkinan nilai, ukuran perolehan informasi memberikan indikasi yang tidak tepat tentang kegunaan atribut tersebut. Dalam kasus ekstrem, atribut seperti *ExactTime* memiliki nilai yang berbeda untuk setiap contoh, yang berarti setiap subset contoh adalah singleton dengan klasifikasi unik, dan ukuran perolehan informasi akan memiliki nilai tertinggi untuk atribut ini. Namun, memilih pemisahan ini terlebih dahulu kemungkinan tidak akan menghasilkan pohon terbaik. Salah satu solusinya adalah menggunakan **gain rasio**. Kemungkinan lain adalah dengan mengizinkan uji Boolean dalam bentuk $A = v_k$, yaitu, memilih hanya satu dari kemungkinan nilai untuk suatu atribut, membiarkan nilai yang tersisa untuk kemungkinan diuji nanti di pohon.
- **Atribut masukan bernilai kontinu dan integer:** Atribut bernilai kontinu atau integer seperti *Tinggi* dan *Berat*, memiliki himpunan kemungkinan nilai yang tak terhingga. Alih-alih menghasilkan cabang yang tak terhingga banyaknya, algoritma pembelajaran pohon keputusan biasanya menemukan **titik pemisahan** yang memberikan perolehan informasi tertinggi. Misalnya, pada simpul tertentu di pohon, pengujian pada $Weight > 160$ mungkin memberikan informasi paling banyak. Terdapat metode yang efisien untuk menemukan titik pisah yang baik: mulailah dengan mengurutkan nilai atribut, lalu pertimbangkan hanya titik pisah yang berada di antara dua contoh dalam urutan terurut yang memiliki klasifikasi berbeda, sambil tetap melacak total berjalan dari contoh positif dan negatif di setiap sisi titik pisah. Pemisahan merupakan bagian termahal dari aplikasi pembelajaran pohon keputusan di dunia nyata.

- **Atribut keluaran bernilai kontinu:** Jika kita mencoba memprediksi nilai keluaran numerik, seperti harga apartemen, maka kita memerlukan **pohon regresi**, bukan pohon klasifikasi. Pohon regresi memiliki fungsi linier pada setiap daunnya dari beberapa subset atribut numerik, alih-alih satu nilai tunggal. Misalnya, cabang untuk apartemen dua kamar tidur mungkin berakhir dengan fungsi linier dari luas persegi, jumlah kamar mandi, dan pendapatan rata-rata untuk lingkungan tersebut. Algoritma pembelajaran harus memutuskan kapan harus berhenti melakukan pemisahan dan mulai menerapkan regresi linier pada atribut-atribut tersebut.

Sistem pembelajaran pohon keputusan untuk aplikasi dunia nyata harus mampu menangani semua masalah ini. Penanganan variabel bernilai kontinu sangat penting, karena proses fisik dan finansial menghasilkan data numerik. Beberapa paket komersial telah dibangun yang memenuhi kriteria ini, dan telah digunakan untuk mengembangkan ribuan sistem lapangan. Di banyak bidang industri dan perdagangan, pohon keputusan biasanya merupakan metode pertama yang dicoba ketika suatu metode klasifikasi akan diekstraksi dari suatu kumpulan data. Salah satu sifat penting dari pohon keputusan adalah memungkinkan manusia untuk memahami alasan keluaran algoritma pembelajaran. (Memang, ini *merupakan persyaratan hukum* untuk keputusan finansial yang tunduk pada undang-undang antidiskriminasi.) Sifat ini tidak dimiliki oleh representasi lain, seperti jaringan saraf.

9.4 MENGEVALUASI DAN MEMILIH HIPOTESIS TERBAIK

Kita ingin mempelajari hipotesis yang paling sesuai dengan data masa depan. Untuk membuatnya lebih tepat, kita perlu mendefinisikan "data masa depan" dan "terbaik". Kita membuat **asumsi stasioneritas**: bahwa terdapat distribusi probabilitas atas contoh-contoh yang tetap stasioner seiring waktu. Setiap titik data contoh (sebelum kita melihatnya) adalah variabel acak E_j yang nilai teramatinya $e_j = (x_j, y_j)$ diambil sampelnya dari distribusi tersebut, dan independen dari contoh-contoh sebelumnya:

$$P(E_j | E_{j-1}, E_{j-2}, \dots) = P(E_j),$$

dan setiap contoh memiliki distribusi probabilitas sebelumnya yang identik:

$$P(E_j) = P(E_{j-1}) = P(E_{j-2}) = \dots.$$

Contoh-contoh yang memenuhi asumsi ini disebut *independen dan terdistribusi identik* atau **i.i.d.** Asumsi i.i.d. menghubungkan masa lalu dengan masa depan; tanpa hubungan semacam itu, semua taruhan batal—masa depan bisa apa saja. (Kita akan melihat nanti bahwa pembelajaran masih dapat terjadi jika ada perubahan lambat dalam distribusi.)

Langkah selanjutnya adalah mendefinisikan "kesesuaian terbaik". Kita mendefinisikan **tingkat kesalahan** suatu hipotesis sebagai proporsi kesalahan yang dibuatnya—proporsi berapa kali $h(x) \neq y$ untuk contoh (x, y) . Nah, hanya karena hipotesis h memiliki tingkat

kesalahan yang rendah pada set pelatihan tidak berarti hipotesis tersebut akan tergeneralisasi dengan baik. Seorang profesor tahu bahwa ujian tidak akan mengevaluasi mahasiswa secara akurat jika mereka telah melihat soal ujian tersebut. Demikian pula, untuk mendapatkan evaluasi hipotesis yang akurat, kita perlu mengujinya pada serangkaian contoh yang belum pernah dilihatnya. Pendekatan paling sederhana adalah yang telah kita lihat: membagi data yang tersedia secara acak menjadi set pelatihan tempat algoritma pembelajaran menghasilkan h dan set pengujian tempat akurasi h dievaluasi. Metode ini, yang terkadang disebut **holdout cross-validation**, memiliki kelemahan karena gagal menggunakan semua data yang tersedia; jika kita menggunakan separuh data untuk set pengujian, maka kita hanya melatih separuh data tersebut, dan kita mungkin mendapatkan hipotesis yang buruk. Di sisi lain, jika kita hanya menyisakan 10% data untuk set pengujian, maka kita mungkin, secara kebetulan statistik, mendapatkan estimasi akurasi aktual yang buruk.

Kita dapat memeras lebih banyak data dan tetap mendapatkan estimasi akurat menggunakan teknik yang disebut **k -fold cross-validation**. Idennya adalah bahwa setiap contoh memiliki fungsi ganda—sebagai data pelatihan dan data uji. Pertama, kita membagi data menjadi k subset yang sama besar. Kemudian, kita melakukan k putaran pembelajaran; pada setiap putaran, $1/k$ data disimpan sebagai set pengujian dan contoh yang tersisa digunakan sebagai data pelatihan. Skor rata-rata set pengujian dari k putaran tersebut seharusnya merupakan estimasi yang lebih baik daripada skor tunggal. Nilai k yang populer adalah 5 dan 10—cukup untuk memberikan estimasi yang secara statistik kemungkinan akurat, dengan biaya waktu komputasi 5 hingga 10 kali lebih lama. Nilai ekstremnya adalah $k = n$, yang juga dikenal sebagai validasi silang **leave-one-out** atau **LOOCV**.

Terlepas dari upaya terbaik para ahli metodologi statistik, pengguna sering kali membatalkan hasil mereka dengan cara tidak sengaja **mengintip** data uji. Mengintip dapat terjadi seperti ini: Algoritma pembelajaran memiliki berbagai "tombol" yang dapat diputar untuk menyesuaikan perilakunya—misalnya, berbagai kriteria berbeda untuk memilih atribut berikutnya dalam pembelajaran pohon keputusan. Peneliti menghasilkan hipotesis untuk berbagai pengaturan tombol, mengukur tingkat kesalahannya pada set uji, dan melaporkan tingkat kesalahan hipotesis terbaik. Sayangnya, mengintip telah terjadi! Alasannya adalah hipotesis tersebut dipilih *berdasarkan tingkat kesalahan set uji*, sehingga informasi tentang set uji telah bocor ke dalam algoritma pembelajaran.

Mengintip adalah konsekuensi dari penggunaan kinerja set uji untuk *memilih* hipotesis dan *mengevaluasinya*. Cara untuk menghindari hal ini adalah dengan *benar-benar* menahan set uji—menguncinya hingga Anda benar-benar selesai belajar dan hanya ingin mendapatkan evaluasi independen dari hipotesis akhir. (Dan kemudian, jika Anda tidak menyukai hasilnya ... Anda harus mendapatkan, dan mengunci, set pengujian yang benar-benar baru jika Anda ingin kembali dan menemukan hipotesis yang lebih baik.) Jika set pengujian terkunci, tetapi Anda masih ingin mengukur kinerja pada data yang tidak terlihat sebagai cara untuk memilih hipotesis yang baik, maka bagilah data yang tersedia (tanpa set pengujian) menjadi set pelatihan dan **set validasi**. Bagian selanjutnya menunjukkan cara menggunakan set validasi untuk menemukan keseimbangan yang baik antara kompleksitas hipotesis dan kesesuaian.

Pemilihan model: Kompleksitas versus kesesuaian

Pada Gambar 9.1, kami menunjukkan bahwa polinomial dengan derajat yang lebih tinggi dapat menyesuaikan data pelatihan dengan lebih baik, tetapi ketika derajatnya terlalu tinggi, polinomial tersebut akan mengalami overfitting, dan berkinerja buruk pada data validasi. Memilih derajat polinomial merupakan contoh dari masalah **pemilihan model**. Anda dapat menganggap tugas menemukan hipotesis terbaik sebagai dua tugas: pemilihan model mendefinisikan ruang hipotesis, dan kemudian **optimasi** menemukan hipotesis terbaik dalam ruang tersebut.

Di bagian ini, kami menjelaskan cara memilih di antara model-model yang diparameterisasi berdasarkan ukuran. Misalnya, dengan polinomial, kita memiliki *ukuran* = 1 untuk fungsi linear, *ukuran* = 2 untuk kuadrat, dan seterusnya. Untuk pohon keputusan, ukurannya bisa berupa jumlah simpul dalam pohon. Dalam semua kasus, kita ingin menemukan nilai parameter *ukuran* yang paling menyeimbangkan underfitting dan overfitting untuk memberikan akurasi set uji terbaik.

Algoritma untuk melakukan pemilihan dan optimasi model ditunjukkan pada Gambar 9.8. Ini adalah pembungkus yang menggunakan algoritma pembelajaran sebagai argumen (misalnya, DECISION-TREE-LEARNING). Pembungkus ini menghitung model berdasarkan parameter, yaitu *ukuran*. Untuk setiap ukuran, pembungkus ini menggunakan validasi silang pada *Learner* untuk menghitung tingkat kesalahan rata-rata pada set pelatihan dan pengujian. Kita mulai dengan model terkecil dan paling sederhana (yang kemungkinan kurang sesuai dengan data), dan melakukan iterasi, dengan mempertimbangkan model yang lebih kompleks di setiap langkah, hingga model mulai mengalami overfit. Pada Gambar 9.9, kita melihat kurva-kurva tipikal: kesalahan set pelatihan menurun secara monoton (meskipun secara umum mungkin terdapat sedikit variasi acak), sementara kesalahan set validasi menurun pada awalnya, dan kemudian meningkat ketika model mulai mengalami overfit. Prosedur validasi silang memilih nilai *ukuran* dengan kesalahan set validasi terendah; titik terendah dari kurva berbentuk U. Kemudian, kita menghasilkan hipotesis dengan *ukuran* tersebut, menggunakan semua data (tanpa menyimpan data apa pun). Akhirnya, tentu saja, kita harus mengevaluasi hipotesis yang dikembalikan pada set pengujian yang terpisah.

```

function CROSS – VALIDATION – WRAPPER(Learner, k, examples) returns a hypothesis

  local variables: errT, an array, indexed by size, storing training – set error rates
                   errV, an array, indexed by size, storing validation – set error rates
  for size = 1 to  $\infty$  do
    errT[size], errV [size]  $\leftarrow$  CROSS – VALIDATION(Learner, size, k, examples)
    if errT has converged then do
      best size  $\leftarrow$  the value of size with minimum errV [size]
      return Learner(best size, examples)

function CROSS – VALIDATION(Learner, size, k, examples) returns two values:
  average training set error rate, average validation set error rate

  fold errT  $\leftarrow$  0; fold errV  $\leftarrow$  0
  for fold = 1 to k do
    training set, validation set  $\leftarrow$  PARTITION(examples, fold, k)
    h  $\leftarrow$  Learner(size, training set)
    fold errT  $\leftarrow$  fold errT + ERROR – RATE(h, training set)
    fold errV  $\leftarrow$  fold errV + ERROR – RATE(h, validation set)
  return fold errT/k, fold errV /k

```

Gambar 9.8 Algoritme untuk memilih model yang memiliki tingkat kesalahan terendah pada data validasi dengan membangun model dengan kompleksitas yang meningkat, dan memilih model dengan tingkat kesalahan empiris terbaik pada data validasi. Di sini, *errT* berarti tingkat kesalahan pada data pelatihan, dan *errV* berarti tingkat kesalahan pada data validasi. *Learner* (*size*, *examples*) mengembalikan hipotesis yang kompleksitasnya ditentukan oleh parameter *size*, dan yang dilatih pada *contoh – contoh* tersebut. PARTITION(*examples*, *fold*, *k*) membagi *contoh* menjadi dua subset: set validasi berukuran N/k dan set pelatihan dengan semua contoh lainnya. Pembagian ini berbeda untuk setiap nilai lipatan.

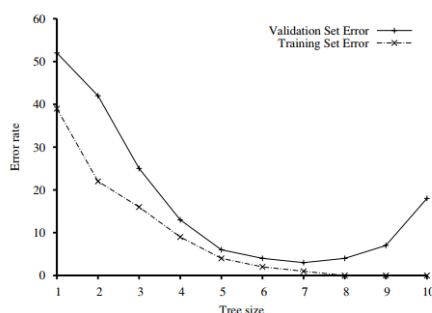
Pendekatan ini mengharuskan algoritma pembelajaran menerima parameter, yaitu *ukuran*, dan menyampaikan hipotesis dengan ukuran tersebut. Seperti yang telah kami sebutkan, untuk pembelajaran pohon keputusan, ukurannya bisa berupa jumlah simpul. Kita dapat memodifikasi DECISION-TREE-LEARNER sehingga ia mengambil jumlah simpul sebagai input, membangun pohon berdasarkan lebarnya terlebih dahulu, bukan kedalamannya terlebih dahulu (tetapi pada setiap tingkat, ia tetap memilih atribut perolehan tertinggi terlebih dahulu), dan berhenti ketika mencapai jumlah simpul yang diinginkan.

Dari tingkat kesalahan ke kerugian

Sejauh ini, kita telah mencoba meminimalkan tingkat kesalahan. Hal ini jelas lebih baik daripada memaksimalkan tingkat kesalahan, tetapi bukan itu inti permasalahannya. Pertimbangkan masalah mengklasifikasikan pesan email sebagai spam atau non-spam. Mengklasifikasikan non-spam sebagai spam (dan dengan demikian berpotensi melewatkan pesan penting) jauh lebih buruk daripada mengklasifikasikan spam sebagai non-spam (dan dengan demikian mengalami gangguan selama beberapa detik). Jadi, pengklasifikasi dengan

tingkat kesalahan 1%, di mana hampir semua kesalahan mengklasifikasikan spam sebagai non-spam, akan lebih baik daripada pengklasifikasi dengan tingkat kesalahan hanya 0,5%, jika sebagian besar kesalahan tersebut mengklasifikasikan non-spam sebagai spam. Kita telah melihat di Bab 16 bahwa pengambil keputusan harus memaksimalkan utilitas yang diharapkan, dan utilitas adalah apa yang juga harus dimaksimalkan oleh pembelajar. Dalam pembelajaran mesin, utilitas biasanya diekspresikan melalui **fungsi kerugian**. Fungsi kerugian $L(x, y, \hat{y})$ didefinisikan sebagai jumlah utilitas yang hilang dengan memprediksi $h(x) = \hat{y}$ ketika jawaban yang benar adalah $f(x) = y$:

$$L(x, y, \hat{y}) = \text{Utility}(\text{result of using } y \text{ given an input } x) \\ - \text{Utility}(\text{result of using } \hat{y} \text{ given an input } x)$$



Gambar 9.9 Tingkat kesalahan pada data pelatihan (garis bawah, putus-putus) dan data validasi (garis atas, utuh) untuk pohon keputusan dengan ukuran berbeda. Kami berhenti ketika tingkat kesalahan set pelatihan mencapai asimtot, lalu memilih pohon dengan kesalahan minimal pada set validasi; dalam hal ini, pohon dengan ukuran 7 simpul.

Ini adalah formulasi paling umum dari fungsi kerugian. Seringkali, versi yang disederhanakan digunakan, $L(y, \hat{y})$, yang tidak bergantung pada x . Kita akan menggunakan versi yang disederhanakan untuk sisa bab ini, yang berarti kita tidak bisa mengatakan bahwa salah mengklasifikasikan surat dari Ibu lebih buruk daripada salah mengklasifikasikan surat dari sepupu kita yang menyebalkan, tetapi kita bisa mengatakan bahwa mengklasifikasikan non-spam sebagai spam 10 kali lebih buruk daripada sebaliknya:

$$L(\text{spam}, \text{nospam}) = 1, \quad L(\text{nospam}, \text{spam}) = 10.$$

Perhatikan bahwa $L(y, y)$ selalu nol; menurut definisi, tidak ada kerugian jika tebakan Anda tepat. Untuk fungsi dengan keluaran diskrit, kita dapat menghitung nilai kerugian untuk setiap kemungkinan kesalahan klasifikasi, tetapi kita tidak dapat menghitung semua kemungkinan untuk data bernilai riil. Jika $f(x)$ adalah 137,035999, kita akan cukup puas dengan $h(x) = 137,036$, tetapi seberapa puaskah seharusnya kita? Secara umum, kesalahan kecil lebih baik daripada kesalahan besar; dua fungsi yang mengimplementasikan gagasan tersebut adalah nilai absolut selisih (disebut kerugian L_1), dan kuadrat selisih (disebut kerugian L_2). Jika kita puas dengan gagasan meminimalkan tingkat kesalahan, kita dapat menggunakan fungsi

kerugian $L_{0/1}$, yang memiliki kerugian 1 untuk jawaban yang salah dan sesuai untuk keluaran bernilai diskrit:

$$\begin{aligned} \text{Absolute value loss: } L_1(y, \hat{y}) &= |y - \hat{y}| \\ \text{Squared error loss: } L_2(y, \hat{y}) &= (y - \hat{y})^2 \\ \text{0/1 loss: } L_{0/1}(y, \hat{y}) &= 0 \text{ if } y = \hat{y}, \text{ else } 1 \end{aligned}$$

Agen pembelajaran secara teoritis dapat memaksimalkan utilitas harapannya dengan memilih hipotesis yang meminimalkan kerugian harapan atas semua pasangan input-output yang akan dilihatnya. Tidak ada gunanya membicarakan harapan ini tanpa mendefinisikan distribusi probabilitas sebelumnya, $\mathbf{P}(X, Y)$ atas contoh-contoh. Misalkan $\mathcal{E}$ adalah himpunan semua kemungkinan contoh input-output. Maka **kerugian generalisasi** harapan untuk hipotesis h (terkait dengan fungsi kerugian L) adalah

$$\text{GenLoss}_L(h) = \sum_{(x,y) \in \mathcal{E}} L(y, h(x))P(x, y),$$

dan hipotesis terbaik, h^* , adalah hipotesis dengan kerugian generalisasi yang diharapkan paling minimum:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{GenLoss}(h)$$

Karena $P(x, y)$ tidak diketahui, agen pembelajaran hanya dapat memperkirakan **kerugian generalisasi** dengan kerugian empiris pada sekumpulan contoh, E :

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{N} \sum_{(x,y) \in E} L(y, h(x)).$$

Hipotesis terbaik yang diestimasi $\hat{h}^*$ adalah hipotesis dengan kerugian empiris minimum:

$$\hat{h}^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \text{EmpLoss}_{L,E}(h)$$

Ada empat alasan mengapa $\hat{h}^*$ mungkin berbeda dari fungsi sebenarnya, f : ketidakterwujudan, varians, noise, dan kompleksitas komputasi. Pertama, f mungkin tidak dapat diwujudkan—mungkin tidak ada di $\mathcal{H}$ —atau mungkin hadir sedemikian rupa sehingga hipotesis lain lebih disukai. Kedua, algoritma pembelajaran akan mengembalikan hipotesis yang berbeda untuk kumpulan contoh yang berbeda, bahkan jika kumpulan tersebut diambil dari fungsi benar yang sama f , dan hipotesis tersebut akan membuat prediksi yang berbeda pada contoh baru. Semakin tinggi varians di antara prediksi, semakin tinggi probabilitas kesalahan yang signifikan. Perhatikan bahwa bahkan ketika masalah dapat diwujudkan, akan

tetap ada varians acak, tetapi varians itu berkurang mendekati nol saat jumlah contoh pelatihan meningkat. Ketiga, f mungkin nondeterministik atau **noisy**—ia dapat mengembalikan nilai yang berbeda untuk $f(x)$ setiap kali x terjadi. Menurut definisi, noise tidak dapat diprediksi; Dalam banyak kasus, hal ini muncul karena label y yang diamati merupakan hasil dari atribut lingkungan yang tidak tercantum dalam x . Dan akhirnya, ketika $\mathcal{H}$ kompleks, pencarian sistematis di seluruh ruang hipotesis dapat menjadi rumit secara komputasi. Hal terbaik yang dapat kita lakukan adalah pencarian lokal (pendakian bukit atau pencarian greedy) yang hanya mengeksplorasi sebagian ruang tersebut. Hal ini menghasilkan kesalahan aproksimasi. Dengan menggabungkan sumber-sumber kesalahan, kita mendapatkan estimasi aproksimasi dari fungsi f yang sebenarnya.

Metode tradisional dalam statistika dan tahun-tahun awal pembelajaran mesin berfokus pada **pembelajaran skala kecil**, dengan jumlah contoh pelatihan berkisar dari puluhan hingga ribuan. Di sini, kesalahan generalisasi sebagian besar berasal dari kesalahan aproksimasi karena tidak memiliki f yang sebenarnya dalam ruang hipotesis, dan dari kesalahan estimasi karena tidak memiliki cukup contoh pelatihan untuk membatasi varians. Dalam beberapa tahun terakhir, terdapat lebih banyak penekanan pada **pembelajaran skala besar**, yang seringkali melibatkan jutaan contoh. Di sini, kesalahan generalisasi didominasi oleh batasan komputasi: terdapat cukup data dan model yang cukup kaya sehingga kita dapat menemukan h yang sangat dekat dengan f yang sebenarnya, tetapi komputasi untuk menemukannya terlalu rumit, sehingga kita memilih pendekatan sub-optimal.

Regularisasi

Pada Bagian 9.4, kita melihat cara melakukan pemilihan model dengan validasi silang pada ukuran model. Pendekatan alternatif adalah mencari hipotesis yang secara langsung meminimalkan jumlah tertimbang kerugian empiris dan kompleksitas hipotesis, yang akan kita sebut biaya total:

$$\begin{aligned} \text{Cost}(h) &= \text{EmpLoss}(h) + \lambda \text{Complexity}(h) \\ \hat{h}^* &= \underset{h \in \mathcal{H}}{\text{argmin}} \text{Cost}(h) \end{aligned}$$

Di sini, λ adalah parameter, bilangan positif yang berfungsi sebagai rasio konversi antara kerugian dan kompleksitas hipotesis (yang bagaimanapun juga tidak diukur pada skala yang sama). Pendekatan ini menggabungkan kerugian dan kompleksitas menjadi satu metrik, yang memungkinkan kita menemukan hipotesis terbaik sekaligus. Sayangnya, kita masih perlu melakukan pencarian validasi silang untuk menemukan hipotesis yang paling baik digeneralisasi, tetapi kali ini dengan nilai λ yang berbeda, bukan *ukuran*. Kita memilih nilai λ yang memberi kita skor set validasi terbaik.

Proses pemberian penalti eksplisit pada hipotesis kompleks ini disebut **regularisasi** (karena mencari fungsi yang lebih teratur, atau kurang kompleks). Perhatikan bahwa fungsi biaya mengharuskan kita membuat dua pilihan: fungsi kerugian dan ukuran kompleksitas, yang disebut fungsi regularisasi. Pilihan fungsi regularisasi bergantung pada ruang hipotesis.

Misalnya, fungsi regularisasi yang baik untuk polinomial adalah jumlah kuadrat koefisien—menjaga jumlah tetap kecil akan menjauhkan kita dari polinomial bergelombang pada Gambar 9.1(b) dan (c). Kami akan menunjukkan contoh jenis regularisasi ini di Bagian 9.6.

Cara lain untuk menyederhanakan model adalah dengan mengurangi dimensi yang digunakan model. Proses **seleksi fitur** dapat dilakukan untuk membuang atribut yang tampaknya tidak relevan. Pemangkasan χ^2 adalah salah satu jenis seleksi fitur.

Sebenarnya, kerugian empiris dan kompleksitas dapat diukur pada skala yang sama, tanpa faktor konversi λ : keduanya dapat diukur dalam bit. Pertama, kodekan hipotesis sebagai program mesin Turing, dan hitung jumlah bitnya. Kemudian, hitung jumlah bit yang diperlukan untuk mengkodekan data, di mana contoh yang diprediksi dengan benar membutuhkan nol bit dan biaya contoh yang diprediksi secara salah bergantung pada seberapa besar kesalahannya. **Panjang deskripsi minimum** atau hipotesis MDL meminimalkan jumlah total bit yang diperlukan. Hal ini berjalan dengan baik pada batasnya, tetapi untuk masalah yang lebih kecil terdapat kesulitan karena pilihan pengkodean untuk program—misalnya, cara terbaik untuk mengkode pohon keputusan sebagai string bit—mempengaruhi hasilnya.

9.5 TEORI PEMBELAJARAN

Pertanyaan utama yang belum terjawab dalam pembelajaran adalah: Bagaimana kita bisa yakin bahwa algoritma pembelajaran kita telah menghasilkan hipotesis yang akan memprediksi nilai yang tepat untuk masukan yang sebelumnya tidak terlihat? Secara formal, bagaimana kita tahu bahwa hipotesis h mendekati fungsi target f jika kita tidak tahu apa itu f ? Pertanyaan-pertanyaan ini telah direnungkan selama beberapa abad. Dalam beberapa dekade terakhir, pertanyaan lain telah muncul: berapa banyak contoh yang kita butuhkan untuk mendapatkan h yang baik? Ruang hipotesis apa yang harus kita gunakan? Jika ruang hipotesis sangat kompleks, dapatkah kita menemukan h terbaik, atau haruskah kita puas dengan nilai maksimum lokal dalam ruang hipotesis? Seberapa kompleks seharusnya h ? Bagaimana kita menghindari overfitting? Bagian ini mengkaji pertanyaan-pertanyaan ini.

Kita akan mulai dengan pertanyaan tentang berapa banyak contoh yang dibutuhkan untuk pembelajaran. Kita melihat dari kurva pembelajaran untuk pembelajaran pohon keputusan pada masalah restoran bahwa peningkatan terjadi dengan lebih banyak data pelatihan. Kurva pembelajaran memang berguna, tetapi spesifik untuk algoritma pembelajaran tertentu pada masalah tertentu. Adakah prinsip umum yang mengatur jumlah contoh yang dibutuhkan secara umum? Pertanyaan seperti ini dibahas oleh **teori pembelajaran komputasional**, yang terletak di persimpangan antara AI, statistika, dan ilmu komputer teoretis. Prinsip dasarnya adalah bahwa *setiap hipotesis yang salah secara serius hampir pasti akan "ditemukan" dengan probabilitas tinggi setelah sejumlah kecil contoh, karena akan menghasilkan prediksi yang salah. Dengan demikian, hipotesis apa pun yang konsisten dengan kumpulan contoh pelatihan yang cukup besar kemungkinan besar tidak akan salah secara serius: artinya, hipotesis tersebut kemungkinan besar akan benar*. Algoritma pembelajaran apa pun yang mengembalikan hipotesis yang kemungkinan besar benar disebut

algoritma **pembelajaran PAC**; kita dapat menggunakan pendekatan ini untuk memberikan batasan pada kinerja berbagai algoritma pembelajaran.

Teorema pembelajaran PAC, seperti semua teorema, merupakan konsekuensi logis dari aksioma. Ketika sebuah *teorema* (berbeda dengan, katakanlah, seorang pakar politik) menyatakan sesuatu tentang masa depan berdasarkan masa lalu, aksioma-aksioma harus menyediakan "inti" untuk membuat hubungan tersebut. Untuk pembelajaran PAC, inti sari disediakan oleh asumsi stasioneritas yang diperkenalkan pada halaman sebelumnya, yang menyatakan bahwa contoh-contoh di masa depan akan diambil dari distribusi tetap yang sama $\mathbf{P}(E) = \mathbf{P}(X, Y)$ dengan contoh-contoh di masa lalu. (Perhatikan bahwa kita tidak perlu tahu distribusi apa itu, hanya saja distribusinya tidak berubah.) Selain itu, untuk menyederhanakannya, kita akan mengasumsikan bahwa fungsi f yang sebenarnya bersifat deterministik dan merupakan anggota kelas hipotesis $\mathcal{H}$ yang sedang dipertimbangkan.

Teorema PAC yang paling sederhana membahas fungsi Boolean, yang kerugian 0/1-nya sesuai. **Tingkat kesalahan** hipotesis h , yang didefinisikan secara informal sebelumnya, didefinisikan secara formal di sini sebagai kesalahan generalisasi yang diharapkan untuk contoh-contoh yang diambil dari distribusi stasioner:

$$\text{error}(h) = \text{GenLoss}_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))P(x, y).$$

Dengan kata lain, $\text{error}(h)$ adalah probabilitas bahwa h salah mengklasifikasikan contoh baru. Besaran ini sama dengan yang diukur secara eksperimental oleh kurva pembelajaran yang ditunjukkan sebelumnya.

Sebuah hipotesis h disebut **approximately correct** jika $\text{error}(h) \leq \epsilon$, di mana ϵ adalah konstanta kecil. Kami akan menunjukkan bahwa kami dapat menemukan N sedemikian rupa sehingga, setelah melihat N contoh, dengan probabilitas tinggi, semua hipotesis yang konsisten akan mendekati benar. Kita dapat menganggap hipotesis yang mendekati benar sebagai "dekat" dengan fungsi sebenarnya dalam ruang hipotesis: hipotesis tersebut terletak di dalam apa yang disebut **bola- ϵ** di sekitar fungsi sebenarnya f . Ruang hipotesis di luar bola ini disebut $\mathcal{H}_{\text{bad}}$.

Kita dapat menghitung probabilitas bahwa hipotesis "salah besar" $h_b \in \mathcal{H}_{\text{bad}}$ konsisten dengan N contoh pertama sebagai berikut. Kita tahu bahwa $\text{error}(h_b) > \epsilon$. Dengan demikian, probabilitas bahwa hipotesis tersebut sesuai dengan contoh yang diberikan paling banyak $1 - \epsilon$. Karena contoh-contoh tersebut independen, batas untuk N contoh adalah

$$P(h_b \text{ agrees with } N \text{ examples}) \leq (1 - \epsilon)^N.$$

Probabilitas bahwa $\mathcal{H}_{\text{bad}}$ mengandung setidaknya satu hipotesis yang konsisten dibatasi oleh jumlah probabilitas individu:

$$P(\mathcal{H}_{\text{bad}} \text{ contains a consistent hypothesis}) \leq |\mathcal{H}_{\text{bad}}|(1 - \epsilon)^N \leq |\mathcal{H}|(1 - \epsilon)^N,$$

di mana kami menggunakan fakta bahwa $|\mathcal{H}_{bad}| \leq |H|$. Kami ingin mengurangi probabilitas kejadian ini di bawah angka kecil δ :

$$|\mathcal{H}|(1 - \epsilon)^N \leq \delta$$

Mengingat $1 - \epsilon \leq e^{-\epsilon}$, kita dapat mencapai hal ini jika kita mengizinkan algoritma untuk melihat

$$N \geq \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + \ln |\mathcal{H}| \right) \quad (9.1)$$

contoh. Dengan demikian, jika suatu algoritma pembelajaran mengembalikan hipotesis yang konsisten dengan banyaknya contoh ini, maka dengan probabilitas setidaknya $1 - \delta$, ia memiliki kesalahan paling banyak ϵ . Dengan kata lain, kemungkinan besar ia mendekati benar. Jumlah contoh yang dibutuhkan, sebagai fungsi dari ϵ dan δ , disebut **kompleksitas sampel** dari ruang hipotesis.

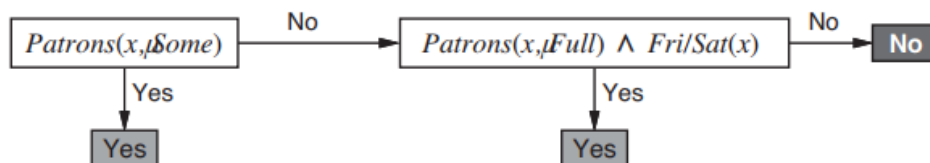
Seperti yang telah kita lihat sebelumnya, jika $\mathcal{H}$ adalah himpunan semua fungsi Boolean pada n atribut, maka $|\mathcal{H}| = 2^n$. Dengan demikian, kompleksitas sampel ruang tersebut bertambah sebesar 2^n . Karena jumlah contoh yang mungkin juga 2^n , hal ini menunjukkan bahwa pembelajaran PAC dalam kelas semua fungsi Boolean memerlukan melihat semua, atau hampir semua, contoh yang mungkin. Pemikiran sejenak mengungkapkan alasannya: $\mathcal{H}$ berisi hipotesis yang cukup untuk mengklasifikasikan himpunan contoh apa pun dengan semua cara yang mungkin. Khususnya, untuk setiap himpunan N contoh, himpunan hipotesis yang konsisten dengan contoh-contoh tersebut berisi jumlah hipotesis yang sama yang memprediksi x_{N+1} positif dan hipotesis yang memprediksi x_{N+1} negatif.

Untuk mendapatkan generalisasi nyata pada contoh-contoh yang belum pernah dilihat, tampaknya kita perlu membatasi ruang hipotesis $\mathcal{H}$ dengan cara tertentu; tetapi tentu saja, jika kita membatasi ruang tersebut, kita mungkin menghilangkan fungsi sebenarnya sama sekali. Ada tiga cara untuk keluar dari dilema ini. Yang pertama, yang akan kita bahas di Bab 19, adalah dengan menggunakan pengetahuan sebelumnya untuk menyelesaikan masalah. Yang kedua, yang telah kita perkenalkan di Bagian 9.4, adalah dengan menekankan bahwa algoritme tidak hanya mengembalikan hipotesis yang konsisten, tetapi lebih disukai yang sederhana (seperti yang dilakukan dalam pembelajaran pohon keputusan). Dalam kasus di mana menemukan hipotesis konsisten yang sederhana dapat dilakukan, hasil kompleksitas sampel umumnya lebih baik daripada analisis yang hanya didasarkan pada konsistensi. Jalan keluar ketiga, yang akan kita bahas selanjutnya, adalah dengan berfokus pada subset yang dapat dipelajari dari keseluruhan ruang hipotesis fungsi Boolean. Pendekatan ini bergantung pada asumsi bahwa bahasa terbatas mengandung hipotesis h yang cukup dekat dengan fungsi sebenarnya f ; manfaatnya adalah ruang hipotesis terbatas memungkinkan generalisasi yang

efektif dan biasanya lebih mudah dicari. Kami sekarang mengkaji salah satu bahasa terbatas tersebut secara lebih rinci.

Contoh pembelajaran PAC: Mempelajari daftar keputusan

Kami sekarang menunjukkan cara menerapkan pembelajaran PAC ke ruang hipotesis baru: **daftar keputusan**. Daftar keputusan terdiri dari serangkaian pengujian, yang masing-masing merupakan konjungsi literal. Jika pengujian berhasil ketika diterapkan pada deskripsi contoh, daftar keputusan menentukan nilai yang akan dikembalikan.



Gambar 9.10 Daftar keputusan untuk masalah restoran.

Jika pengujian gagal, pemrosesan dilanjutkan dengan pengujian berikutnya dalam daftar. Daftar keputusan menyerupai pohon keputusan, tetapi struktur keseluruhannya lebih sederhana: hanya bercabang dalam satu arah. Sebaliknya, pengujian individual lebih kompleks. Gambar 9.10 menunjukkan daftar keputusan yang merepresentasikan hipotesis berikut:

$$WillWait \Leftrightarrow (Patrons = Some) \vee (Patrons = Full \wedge Fri/Sat).$$

Jika kita mengizinkan pengujian dengan ukuran sembarang, maka daftar keputusan dapat merepresentasikan fungsi Boolean apa pun. Di sisi lain, jika kita membatasi ukuran setiap pengujian hingga maksimal k literal, maka algoritma pembelajaran dapat berhasil melakukan generalisasi dari sejumlah kecil contoh. Kita menyebut bahasa ini k_{DL} . Contoh pada Gambar 18.10 berada dalam 2-DL. Mudah untuk menunjukkan bahwa k_{DL} mencakup sebagai subset bahasa k_{DT} , himpunan semua pohon keputusan dengan kedalaman maksimal k . Penting untuk diingat bahwa bahasa tertentu yang dirujuk oleh k_{DL} bergantung pada atribut yang digunakan untuk mendeskripsikan contoh. Kita akan menggunakan notasi $k_{DL}(n)$ untuk menyatakan bahasa k_{DL} menggunakan n atribut Boolean.

Tugas pertama adalah menunjukkan bahwa k_{DL} dapat dipelajari—yaitu, bahwa fungsi apa pun dalam k_{DL} dapat didekati secara akurat setelah pelatihan pada sejumlah contoh yang wajar. Untuk melakukan ini, kita perlu menghitung jumlah hipotesis dalam bahasa tersebut. Misalkan bahasa pengujian—konjungsi maksimal k literal dengan n atribut—adalah $Conj(n, k)$. Karena daftar keputusan dibangun dari pengujian, dan karena setiap pengujian dapat dikaitkan dengan hasil *Yes* atau *No*, atau dapat dihilangkan dari daftar keputusan, terdapat maksimal $3^{|Conj(n, k)|}$ himpunan pengujian komponen yang berbeda. Setiap himpunan pengujian ini dapat berupa urutan apa pun, sehingga

$$|k_{DL}(n)| \leq 3^{|Conj(n, k)|} |Conj(n, k)|!$$

Jumlah konjungsi k literal dari n atribut diberikan oleh

$$|\text{Conj}(n, k)| = \sum_{i=0}^k \binom{2n}{i} = O(n^k).$$

Oleh karena itu, setelah beberapa pekerjaan, kita memperoleh

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}.$$

Kita dapat memasukkan ini ke Persamaan (9.1) untuk menunjukkan bahwa jumlah contoh yang dibutuhkan untuk pembelajaran PAC fungsi $k\text{-DL}$ adalah polinomial dalam n :

$$N \geq \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + O(n^k \log_2(n^k)) \right).$$

Oleh karena itu, setiap algoritma yang mengembalikan daftar keputusan yang konsisten akan melakukan PAC-learn pada fungsi $k\text{-DL}$ dalam jumlah contoh yang wajar, untuk k yang kecil.

Tugas selanjutnya adalah menemukan algoritma efisien yang mengembalikan daftar keputusan yang konsisten. Kita akan menggunakan algoritma greedy yang disebut DECISION-LIST-LEARNING yang berulang kali menemukan pengujian yang sesuai persis dengan beberapa subset dari set pelatihan. Setelah menemukan pengujian tersebut, algoritma tersebut menambahkannya ke daftar keputusan yang sedang dibangun dan menghapus contoh yang sesuai. Kemudian algoritma tersebut membangun sisa daftar keputusan, hanya menggunakan contoh yang tersisa. Hal ini diulang hingga tidak ada contoh yang tersisa. Algoritma ini ditunjukkan pada Gambar 9.11.

function DECISION – LIST – LEARNING(*examples*) **returns** a decision list, or *failure*

if *examples* is empty **then return** the trivial decision list *No*

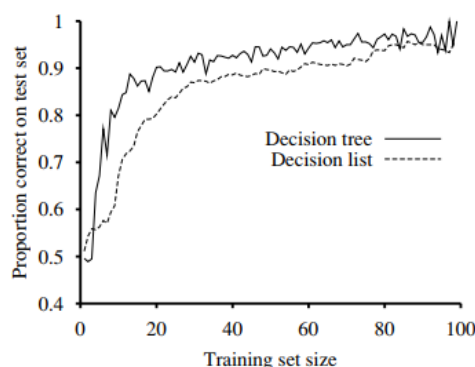
$t \leftarrow$ a test that matches a nonempty subset $examples_t$ of *examples*
such that the members of $examples_t$ are all positive or all negative

if there is no such t **then return** *failure*

if the examples in $examples_t$ are positive **then** $o \leftarrow \text{Yes}$ **else** $o \leftarrow \text{No}$

return a decision list with initial test t and outcome o and remaining tests given by
DECISION – LIST – LEARNING($examples - examples_t$)

Gambar 9.11 Algoritma untuk mempelajari daftar keputusan.

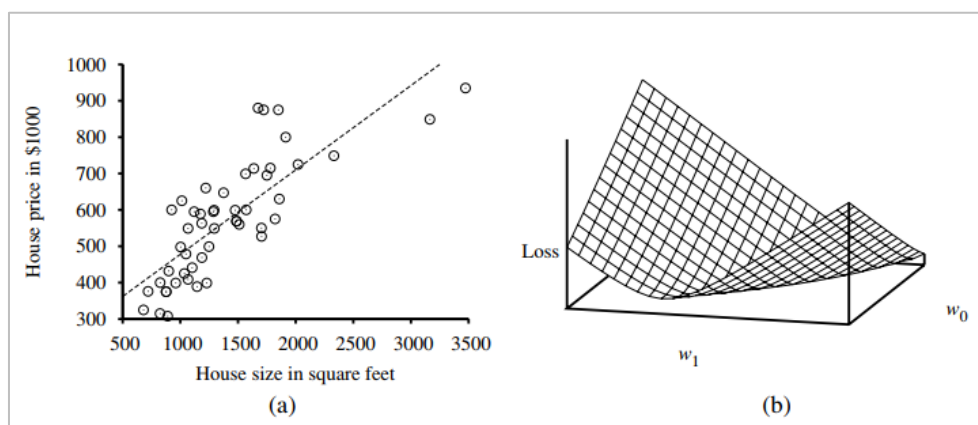


Gambar 9.12 Kurva pembelajaran untuk algoritma DECISION-LIST-LEARNING pada data restoran. Kurva untuk DECISION-TREE-LEARNING ditampilkan sebagai perbandingan.

Algoritma ini tidak menentukan metode untuk memilih pengujian berikutnya yang akan ditambahkan ke daftar keputusan. Meskipun hasil formal yang diberikan sebelumnya tidak bergantung pada metode pemilihan, tampaknya masuk akal untuk lebih memilih pengujian kecil yang cocok dengan kumpulan besar contoh yang diklasifikasikan secara seragam, sehingga keseluruhan daftar keputusan akan sepadat mungkin. Strategi paling sederhana adalah menemukan nilai uji t terkecil yang cocok dengan setiap subset yang terklasifikasi secara seragam, terlepas dari ukuran subset tersebut. Pendekatan ini pun cukup efektif, seperti yang ditunjukkan pada Gambar 9.12.

9.6 REGRESI DAN KLASIFIKASI DENGAN MODEL LINIER

Sekarang saatnya beralih dari pohon keputusan dan daftar ke ruang hipotesis yang berbeda, yang telah digunakan selama ratusan tahun: kelas **fungsi linier** dengan input bernilai kontinu. Kita akan mulai dengan kasus paling sederhana: regresi dengan fungsi linier univariat, atau dikenal sebagai "pencocokan garis lurus". Bagian ini membahas kasus multivariat. Bagian selanjutnya menunjukkan cara mengubah fungsi linier menjadi pengklasifikasi dengan menerapkan ambang batas keras dan lunak.



Gambar 9.13 (a) Titik data harga versus luas lantai rumah yang dijual di Berkeley, CA, pada bulan Juli 2009, beserta hipotesis fungsi linear yang meminimalkan kerugian akibat kesalahan kuadrat: $y = 0,232x + 246$. (b) Plot fungsi kerugian $\sum_j (w_1 x_j + w_0 - y_j)^2$ untuk

berbagai nilai w_0, w_1 . Perhatikan bahwa fungsi kerugian tersebut cembung, dengan satu minimum global.

Regresi Linier Univariat

Fungsi linier univariat (garis lurus) dengan masukan x dan keluaran y memiliki bentuk $y = w_1x + w_0$, di mana w_0 dan w_1 adalah koefisien bernilai riil yang harus dipelajari. Kita menggunakan huruf w karena kita menganggap koefisien sebagai **berat**; nilai y diubah dengan mengubah bobot relatif suatu suku atau suku lainnya. Kita akan mendefinisikan $\mathbf{w}$ sebagai vektor $[w_0, w_1]$, dan mendefinisikan

$$h_w(x) = w_1x + w_0.$$

Gambar 9.13(a) menunjukkan contoh himpunan pelatihan n titik pada bidang x, y , yang masing-masing titiknya mewakili ukuran dalam kaki persegi dan harga rumah yang ditawarkan untuk dijual. Tugas menemukan h_w yang paling sesuai dengan data ini disebut **regresi linier**. Untuk menyesuaikan garis dengan data, yang perlu kita lakukan hanyalah menemukan nilai bobot $[w_0, w_1]$ yang meminimalkan kerugian empiris. Secara tradisional (berasal dari Gauss) menggunakan fungsi kerugian kuadrat, L_2 , yang dijumlahkan dari semua contoh pelatihan:

$$\text{Loss}(h_w) = \sum_{j=1}^N L_2(y_j, h_w(x_j)) = \sum_{j=1}^N (y_j - h_w(x_j))^2 = \sum_{j=1}^N (y_j - (w_1x_j + w_0))^2.$$

Kita ingin mencari $\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \text{Loss}(h_w)$. Jumlah $\sum_{j=1}^N (y_j - (w_1x_j + w_0))^2$ diminimalkan ketika turunan parsialnya terhadap w_0 dan w_1 adalah nol:

$$\frac{\partial}{\partial w_0} \sum_{j=1}^N (y_j - (w_1x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1} \sum_{j=1}^N (y_j - (w_1x_j + w_0))^2 = 0. \quad (9.2)$$

Persamaan ini memiliki solusi unik:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = \left(\sum y_j - w_1 \left(\sum x_j \right) \right) / N. \quad (9.3)$$

Untuk contoh pada Gambar 9.13(a), solusinya adalah $w_1 = 0,232$, $w_0 = 246$, dan garis dengan bobot tersebut ditunjukkan sebagai garis putus-putus pada gambar.

Banyak bentuk pembelajaran melibatkan penyesuaian bobot untuk meminimalkan kerugian, jadi akan membantu untuk memiliki gambaran mental tentang apa yang terjadi di **ruang bobot**—ruang yang didefinisikan oleh semua kemungkinan pengaturan bobot. Untuk regresi linier univariat, ruang bobot yang didefinisikan oleh w_0 dan w_1 adalah dua dimensi,

sehingga kita dapat membuat grafik kerugian sebagai fungsi dari w_0 dan w_1 dalam plot 3D (lihat Gambar 9.13(b)). Kita melihat bahwa fungsi kerugian bersifat **cembung**, seperti yang didefinisikan pada halaman sebelumnya; ini berlaku untuk *setiap* masalah regresi linier dengan fungsi kerugian L_2 , dan menyiratkan bahwa tidak ada minimum lokal. Dalam beberapa hal, itulah akhir cerita untuk model linier; Jika kita perlu menyesuaikan garis dengan data, kita menerapkan Persamaan (9.3).

Untuk melampaui model linear, kita perlu menghadapi kenyataan bahwa persamaan yang mendefinisikan kerugian minimum (seperti dalam Persamaan (9.2)) seringkali tidak memiliki solusi bentuk tertutup. Sebaliknya, kita akan menghadapi masalah pencarian optimasi umum dalam ruang bobot kontinu. Sebagaimana ditunjukkan di Bagian 4., masalah tersebut dapat diatasi dengan algoritma pendakian bukit yang mengikuti **gradien** fungsi yang akan dioptimalkan. Dalam hal ini, karena kita mencoba meminimalkan kerugian, kita akan menggunakan **penurunan gradien**. Kita memilih titik awal mana pun dalam ruang bobot—di sini, sebuah titik pada bidang (w_0, w_1) —lalu bergerak ke titik tetangga yang menurun, mengulanginya hingga kita konvergen pada kerugian minimum yang mungkin:

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

for each w_i in $\mathbf{w}$ **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) \quad (9.4)$$

Parameter α , yang kita sebut **ukuran langkah** di Bagian 4.2, biasanya disebut **laju pembelajaran** ketika kita mencoba meminimalkan kerugian dalam suatu permasalahan pembelajaran. Laju ini bisa berupa konstanta tetap, atau dapat menurun seiring waktu seiring berjalannya proses pembelajaran.

Untuk regresi univariat, fungsi kerugian adalah fungsi kuadrat, sehingga turunan parsialnya akan berupa fungsi linear. (Satu-satunya kalkulus yang perlu Anda ketahui adalah $\frac{\partial}{\partial x} x^2 = 2x$ dan $\frac{\partial}{\partial x} x = 1$.) Mari kita hitung turunan parsial—kemiringannya—dalam kasus sederhana hanya dengan satu contoh pelatihan, (x, y) :

$$\begin{aligned} \frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)), \end{aligned} \quad (18.5)$$

menerapkan ini pada w_0 dan w_1 kita mendapatkan:

$$\frac{\partial}{\partial w_0} \text{Loss}(w) = -2(y - h_w(x)); \quad \frac{\partial}{\partial w_1} \text{Loss}(w) = -2(y - h_w(x)) \times x$$

Kemudian, dengan memasukkan kembali ini ke Persamaan (9.4), dan melipat angka 2 ke dalam laju pembelajaran yang tidak ditentukan α , kita memperoleh aturan pembelajaran berikut untuk bobot:

$$w_0 \leftarrow w_0 + \alpha(y - h_w(x)); \quad w_1 \leftarrow w_1 + \alpha(y - h_w(x)) \times x$$

Pembaruan ini masuk akal secara intuitif: jika $h_w(x) > y$, yaitu, keluaran hipotesis terlalu besar, kurangi w_0 sedikit, dan kurangi w_1 jika x merupakan masukan positif, tetapi tingkatkan w_1 jika x merupakan masukan negatif.

Persamaan sebelumnya mencakup satu contoh pelatihan. Untuk N contoh pelatihan, kita ingin meminimalkan jumlah kerugian individual untuk setiap contoh. Turunan dari suatu jumlah adalah jumlah turunannya, sehingga kita peroleh:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) \times x_j.$$

Pembaruan ini merupakan aturan pembelajaran **penurunan gradien batch** untuk regresi linier univariat. Konvergensi ke minimum global yang unik dijamin (selama kita memilih α yang cukup kecil) tetapi mungkin sangat lambat: kita harus menggilir semua data pelatihan untuk setiap langkah, dan mungkin ada banyak langkah.

Ada kemungkinan lain, yang disebut **penurunan gradien stokastik**, di mana kita hanya mempertimbangkan satu titik pelatihan pada satu waktu, mengambil satu langkah setelah setiap langkah menggunakan Persamaan (9.5). Penurunan gradien stokastik dapat digunakan dalam pengaturan daring, di mana data baru masuk satu per satu, atau luring, di mana kita menggilir data yang sama sebanyak yang diperlukan, mengambil satu langkah setelah mempertimbangkan setiap contoh tunggal. Seringkali lebih cepat daripada penurunan gradien batch. Namun, dengan laju pembelajaran tetap α , hal itu tidak menjamin konvergensi; ia dapat berosilasi di sekitar minimum tanpa menetap. Dalam beberapa kasus, seperti yang akan kita lihat nanti, jadwal laju pembelajaran yang menurun (seperti dalam simulasi anil) memang menjamin konvergensi.

Regresi linier multivariat

Kita dapat dengan mudah memperluasnya ke masalah **regresi linier multivariat**, di mana setiap contoh $\mathbf{x}_j$ adalah vektor berelemen n . Ruang hipotesis kita adalah himpunan fungsi dalam bentuk

$$h_{sw}(x_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

Suku w_0 , intersep, tampak berbeda dari yang lain. Kita dapat memperbaikinya dengan menciptakan atribut masukan semu, $x_{j,0}$, yang didefinisikan selalu sama dengan 1. Maka, h hanyalah hasil perkalian titik dari bobot dan vektor masukan (atau ekuivalennya, hasil perkalian matriks dari transpos bobot dan vektor masukan):

$$h_{sw}(x_j) = w \cdot x_j = w^T x_j = \sum_i w_i x_{j,i}.$$

Vektor bobot terbaik, w^* , meminimalkan kerugian kesalahan kuadrat pada contoh-contoh berikut:

$$w^* = \underset{w}{\operatorname{argmin}} \sum_j L_2(y_j, w \cdot x_j).$$

Regresi linier multivariat sebenarnya tidak jauh lebih rumit daripada kasus univariat yang baru saja kita bahas. Penurunan gradien akan mencapai minimum (unik) dari fungsi kerugian; persamaan pembaruan untuk setiap bobot w_i adalah

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i} (y_j - h_w(x_j)). \quad (9.6)$$

Dimungkinkan juga untuk menyelesaikan secara analitis untuk w yang meminimalkan kerugian. Misalkan y adalah vektor keluaran untuk contoh pelatihan, dan X adalah **matriks data**, yaitu matriks masukan dengan satu contoh n -dimensi per baris. Maka solusinya

$$w^* = (X^T X)^{-1} X^T y$$

meminimalkan galat kuadrat.

Dengan regresi linier univariat, kita tidak perlu khawatir tentang overfitting. Namun, dengan regresi linier multivariat dalam ruang berdimensi tinggi, ada kemungkinan beberapa dimensi yang sebenarnya tidak relevan muncul secara kebetulan menjadi berguna, sehingga mengakibatkan **overfitting**.

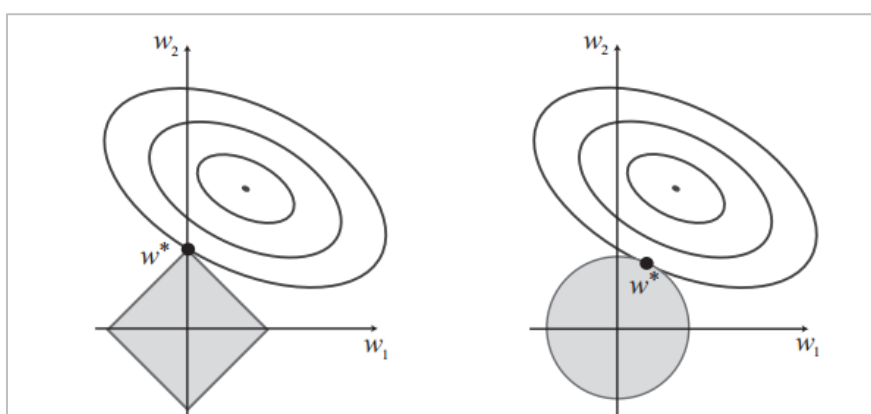
Oleh karena itu, **regularisasi** umum digunakan pada fungsi linier multivariat untuk menghindari overfitting. Ingatlah bahwa dengan regularisasi, kita meminimalkan total biaya hipotesis, dengan menghitung kerugian empiris dan kompleksitas hipotesis:

$$\text{Cost}(h) = \text{EmpLoss}(h) + \lambda \text{Complexity}(h).$$

Untuk fungsi linear, kompleksitas dapat ditentukan sebagai fungsi bobot. Kita dapat mempertimbangkan keluarga fungsi regularisasi:

$$\text{Complexity}(h_w) = L_q(w) = \sum_i |w_i|^q.$$

Seperti halnya fungsi kerugian,⁶ dengan $q = 1$ kita memiliki regularisasi L_1 , yang meminimalkan jumlah nilai absolut; dengan $q = 2$, regularisasi L_2 meminimalkan jumlah kuadrat. Fungsi regularisasi mana yang harus Anda pilih? Hal itu bergantung pada masalah spesifiknya, tetapi regularisasi L_1 memiliki keunggulan penting: cenderung menghasilkan **model jarang**. Artinya, sering kali menetapkan banyak bobot ke nol, yang secara efektif menyatakan atribut yang sesuai menjadi tidak relevan—sama seperti yang dilakukan PEMBELAJARAN POHON KEPUTUSAN (meskipun dengan mekanisme yang berbeda). Hipotesis yang membuang atribut dapat lebih mudah dipahami manusia, dan mungkin lebih kecil kemungkinannya untuk overfitting.



Gambar 9.14 Mengapa regularisasi L_1 cenderung menghasilkan model yang jarang. (a) Dengan regularisasi L_1 (kotak), kerugian minimal yang dapat dicapai (kontur konsentris) sering terjadi pada sumbu, yang berarti bobot nol. (b) Dengan regularisasi L_2 (lingkaran), kerugian minimal cenderung terjadi di mana saja pada lingkaran, tidak memberikan preferensi pada bobot nol.

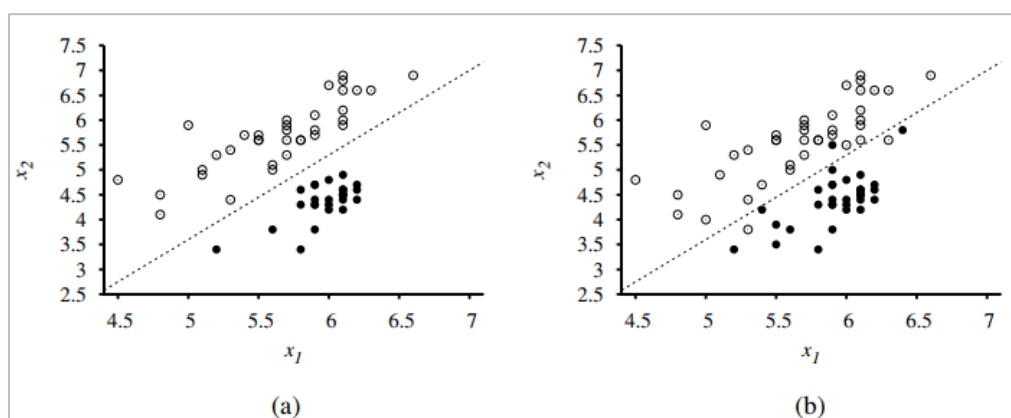
Gambar 9.14 memberikan penjelasan intuitif mengapa regularisasi L_1 mengarah ke bobot nol, sedangkan regularisasi L_2 tidak. Perhatikan bahwa meminimalkan $\text{Loss}(\mathbf{w}) + \lambda \text{Kompleksitas}(\mathbf{w})$ setara dengan meminimalkan $\text{Kerugian}(\mathbf{w})$ dengan batasan bahwa $\text{Kompleksitas}(\mathbf{w}) \leq c$, untuk beberapa konstanta c yang terkait dengan λ . Sekarang, pada Gambar 9.14(a) kotak berbentuk berlian mewakili himpunan titik $\mathbf{w}$ dalam ruang bobot dua dimensi yang memiliki kompleksitas L_1 kurang dari c ; solusi kita harus berada di suatu tempat di dalam kotak ini. Oval konsentris mewakili kontur fungsi kerugian, dengan kerugian minimum di tengah. Kita ingin menemukan titik di dalam kotak yang paling dekat dengan minimum; Anda dapat melihat dari diagram bahwa, untuk posisi minimum dan konturnya yang sembarang, sudut kotak akan cenderung menemukan jalannya paling dekat dengan minimum, hanya karena sudut-sudutnya runcing. Dan tentu saja, sudut-sudutnya adalah titik-titik yang memiliki nilai nol dalam beberapa dimensi. Pada Gambar 9.14(b), kami melakukan hal yang sama untuk

ukuran kompleksitas L_2 , yang merepresentasikan lingkaran, bukan berlian. Di sini Anda dapat melihat bahwa, secara umum, tidak ada alasan bagi perpotongan untuk muncul pada salah satu sumbu; dengan demikian, regularisasi L_2 cenderung tidak menghasilkan bobot nol. Hasilnya adalah jumlah contoh yang diperlukan untuk menemukan h yang baik bersifat linear dalam jumlah fitur yang tidak relevan untuk regularisasi L_2 , tetapi hanya logaritmik dengan regularisasi L_1 . Bukti empiris pada banyak masalah mendukung analisis ini.

Cara lain untuk melihatnya adalah bahwa regularisasi L_1 menganggap serius sumbu-sumbu dimensi, sementara L_2 memperlakukannya sebagai sembarang. Fungsi L_2 berbentuk bola, yang membuatnya invarian rotasi: Bayangkan sekumpulan titik pada sebuah bidang, diukur berdasarkan koordinat x dan y -nya. Sekarang bayangkan memutar sumbu-sumbu tersebut sebesar 45° . Anda akan mendapatkan sekumpulan nilai (x', y') yang berbeda yang merepresentasikan titik-titik yang sama. Jika Anda menerapkan regularisasi L_2 sebelum dan sesudah rotasi, Anda akan mendapatkan titik yang persis sama dengan jawabannya (meskipun titik tersebut akan dideskripsikan dengan koordinat (x', y') yang baru). Hal ini sesuai ketika pilihan sumbu benar-benar arbitrer—ketika tidak penting apakah dua dimensi Anda adalah jarak utara dan timur; atau jarak timur laut dan tenggara. Dengan regularisasi L_1 , Anda akan mendapatkan jawaban yang berbeda, karena fungsi L_1 tidak invarian rotasi. Hal ini sesuai ketika sumbu-sumbunya tidak dapat dipertukarkan; tidak masuk akal untuk memutar "jumlah kamar mandi" 45° derajat ke arah "ukuran kavling".

Pengklasifikasi Linear dengan Ambang Batas Keras

Fungsi linear dapat digunakan untuk melakukan klasifikasi sekaligus regresi. Sebagai contoh, Gambar 9.15(a) menunjukkan titik data dari dua kelas: gempa bumi (yang menarik bagi seismolog) dan ledakan bawah tanah (yang menarik bagi pakar pengendalian senjata). Setiap titik didefinisikan oleh dua nilai masukan, x_1 dan x_2 , yang merujuk pada magnitudo gelombang tubuh dan gelombang permukaan yang dihitung dari sinyal seismik. Dengan data pelatihan ini, tugas klasifikasi adalah mempelajari hipotesis h yang akan mengambil titik-titik (x_1, x_2) baru dan menghasilkan 0 untuk gempa bumi atau 1 untuk ledakan.



Gambar 9.15 (a) Plot dua parameter data seismik, magnitudo gelombang badan x_1 dan magnitudo gelombang permukaan x_2 , untuk gempa bumi (lingkaran putih) dan ledakan nuklir (lingkaran hitam) yang terjadi antara tahun 1982 dan 1990 di Asia dan Timur Tengah.

Batas keputusan antar kelas juga ditunjukkan. (b) Domain yang sama dengan lebih banyak titik data. Gempa bumi dan ledakan tidak lagi dapat dipisahkan secara linear.

Batas keputusan adalah garis (atau permukaan, dalam dimensi yang lebih tinggi) yang memisahkan dua kelas. Pada Gambar 9.15(a), batas keputusan berupa garis lurus. Batas keputusan linier disebut **pemisah linier**, dan data yang memiliki pemisah tersebut disebut dapat **dipisahkan secara linier**. Pemisah linier dalam kasus ini didefinisikan oleh

$$x_2 = 1.7x_1 - 4.9 \quad \text{atau} \quad -4.9 + 1.7x_1 - x_2 = 0.$$

Ledakan, yang ingin kita klasifikasikan dengan nilai 1, berada di sebelah kanan garis ini dengan nilai x_1 yang lebih tinggi dan nilai x_2 yang lebih rendah, sehingga merupakan titik-titik yang memiliki $-4.9 + 1.7x_1 - x_2 > 0$, sedangkan gempa bumi memiliki $-4.9 + 1.7x_1 - x_2 < 0$. Dengan menggunakan konvensi input dummy $x_0 = 1$, kita dapat menuliskan hipotesis klasifikasi sebagai

$$h_w(x) = 1 \text{ if } w \cdot x \geq 0 \text{ and } 0 \text{ otherwise.}$$

Sebagai alternatif, kita dapat menganggap h sebagai hasil dari melewati fungsi linear $w \cdot x$ melalui **fungsi ambang batas**:

$$h_w(x) = \text{Threshold}(w \cdot x) \text{ where } \text{Threshold}(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise.}$$

Fungsi ambang batas ditunjukkan pada Gambar 9.17(a).

Setelah hipotesis $h_w(x)$ memiliki bentuk matematika yang terdefinisi dengan baik, kita dapat mempertimbangkan pemilihan bobot w untuk meminimalkan kerugian. Pada Bagian sebelumnya, kita melakukan hal ini baik dalam bentuk tertutup (dengan menetapkan gradien ke nol dan menyelesaikannya untuk bobot) maupun dengan penurunan gradien dalam ruang bobot. Di sini, kita tidak dapat melakukan keduanya karena gradiennya nol hampir di mana pun dalam ruang bobot kecuali pada titik-titik di mana $w \cdot x = 0$, dan pada titik-titik tersebut gradiennya tidak terdefinisi.

Namun, terdapat aturan pembaruan bobot sederhana yang konvergen ke suatu solusi—yaitu, pemisah linier yang mengklasifikasikan data dengan sempurna—dengan syarat data tersebut dapat dipisahkan secara linier. Untuk satu contoh (x, y) , kita memiliki

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times x_i \tag{9.7}$$

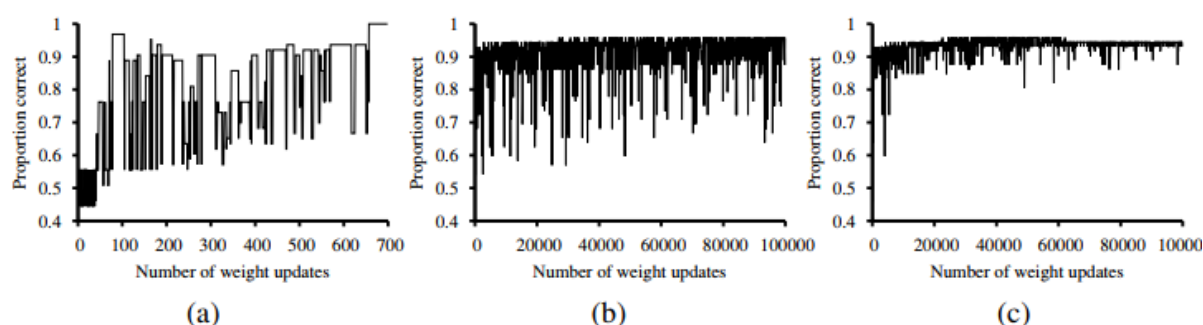
yang pada dasarnya identik dengan Persamaan (9.6), aturan pembaruan untuk regresi linier! Aturan ini disebut **aturan pembelajaran perceptron**, untuk alasan yang akan dijelaskan di Bagian 9.7. Namun, karena kita mempertimbangkan masalah klasifikasi 0/1, perilakunya agak

berbeda. Baik nilai sebenarnya y maupun keluaran hipotesis $h_w(\mathbf{x})$ adalah 0 atau 1, sehingga ada tiga kemungkinan:

- Jika keluarannya benar, yaitu $y = h_w(\mathbf{x})$, maka bobot tidak berubah.
- Jika y adalah 1 tetapi $h_w(\mathbf{x})$ adalah 0, maka w_i meningkat ketika masukan x_i yang bersesuaian positif dan *menurun* ketika x_i negatif. Hal ini masuk akal, karena kita ingin membuat $\mathbf{w} \cdot \mathbf{x}$ lebih besar sehingga $h_w(\mathbf{x})$ menghasilkan 1.
- Jika y adalah 0 tetapi $h_w(\mathbf{x})$ adalah 1, maka w_i menurun ketika masukan x_i yang bersesuaian positif dan *meningkat* ketika x_i negatif. Hal ini masuk akal, karena kita ingin memperkecil $\mathbf{w} \cdot \mathbf{x}$ sehingga $h_w(\mathbf{x})$ menghasilkan 0.

Biasanya, aturan pembelajaran diterapkan satu contoh pada satu waktu, memilih contoh secara acak (seperti dalam penurunan gradien stokastik). Gambar 9.16(a) menunjukkan **kurva pelatihan** untuk aturan pembelajaran ini yang diterapkan pada data gempa bumi/ledakan yang ditunjukkan pada Gambar 9.15(a). Kurva pelatihan mengukur kinerja pengklasifikasi pada set pelatihan tetap saat proses pembelajaran berlangsung pada set pelatihan yang sama. Kurva tersebut menunjukkan aturan pembaruan yang konvergen ke pemisah linier dengan kesalahan nol. Proses "konvergensi" ini tidak sepenuhnya indah, tetapi selalu berhasil. Proses khusus ini membutuhkan 657 langkah untuk konvergen, untuk set data dengan 63 contoh, sehingga setiap contoh disajikan rata-rata sekitar 10 kali. Biasanya, variasi antar proses sangat besar.

Kita telah mengatakan bahwa aturan pembelajaran perceptron konvergen ke pemisah linier sempurna ketika titik-titik data dapat dipisahkan secara linier, tetapi bagaimana jika tidak? Situasi ini terlalu umum di dunia nyata. Misalnya, Gambar 9.15(b) menambahkan kembali titik-titik data yang ditinggalkan oleh Kebeasy dan rekan rekan. (1998) ketika mereka memplot data yang ditunjukkan pada Gambar 9.15(a). Pada Gambar 9.16(b), kami menunjukkan aturan pembelajaran perceptron gagal konvergen bahkan setelah 10.000 langkah: meskipun mencapai solusi kesalahan minimum (tiga kesalahan) berkali-kali, algoritme terus mengubah bobot. Secara umum, aturan perceptron mungkin tidak konvergen ke solusi stabil untuk laju pembelajaran tetap α , tetapi jika α meluruh sebagai $O(1/t)$ di mana t adalah nomor iterasi, maka aturan tersebut dapat ditunjukkan konvergen ke solusi kesalahan minimum ketika contoh disajikan dalam urutan acak.⁷ Dapat juga ditunjukkan bahwa menemukan solusi kesalahan minimum adalah NP-keras, jadi orang berharap bahwa banyak presentasi contoh akan diperlukan agar konvergensi tercapai. Gambar 9.16(b) menunjukkan proses pelatihan dengan jadwal laju pembelajaran $\alpha(t) = 1000/(1000 + t)$: konvergensi tidak sempurna setelah 100.000 iterasi, tetapi jauh lebih baik daripada kasus α tetap.



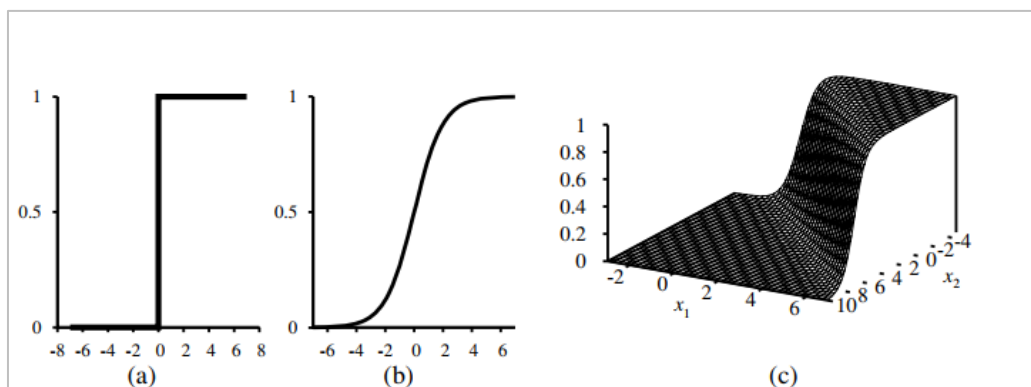
Gambar 9.16 (a) Plot akurasi total set pelatihan vs. jumlah iterasi melalui set pelatihan untuk aturan pembelajaran perceptron, dengan data gempa bumi/ledakan yang diberikan pada Gambar 9.15(a). (b) Plot yang sama untuk data yang berisik dan tidak dapat dipisahkan pada Gambar 9.15(b); perhatikan perubahan skala sumbu x . (c) Plot yang sama seperti pada (b), dengan jadwal laju pembelajaran $\alpha(t) = 1000/(1000 + t)$.

Klasifikasi Linear dengan Regresi Logistik

Kita telah melihat bahwa melewati keluaran fungsi linear melalui fungsi ambang batas menghasilkan pengklasifikasi linear; namun sifat keras ambang batas menyebabkan beberapa masalah: hipotesis $h_w(\mathbf{x})$ tidak terdiferensiasi dan sebenarnya merupakan fungsi diskontinu dari masukan dan bobotnya; hal ini membuat pembelajaran dengan aturan perceptron menjadi petualangan yang sangat tidak terduga. Lebih lanjut, pengklasifikasi linear selalu menghasilkan prediksi 1 atau 0 yang sepenuhnya meyakinkan, bahkan untuk contoh yang sangat dekat dengan batas; dalam banyak situasi, kita benar-benar membutuhkan prediksi yang lebih tergradasi.

Semua masalah ini dapat diselesaikan sebagian besar dengan melunakkan fungsi ambang batas—mengaproksimasi ambang batas keras dengan fungsi kontinu dan terdiferensiasi. Dalam Bab 5 kita melihat dua fungsi yang tampak seperti ambang batas lunak: integral dari distribusi normal baku (digunakan untuk model probit) dan fungsi logistik (digunakan untuk model logit). Meskipun kedua fungsi tersebut sangat mirip bentuknya, fungsi logistik

$$\text{Logistic}(z) = \frac{1}{1 + e^{-z}}$$



Gambar 9.17 (a) Fungsi ambang batas keras *Ambang* (z) dengan keluaran 0/1. Perhatikan bahwa fungsi tersebut tidak terdiferensiasi pada $z = 0$. **(b)** Fungsi logistik, $Logistik(z) = \frac{1}{1+e^{-z}}$, juga dikenal sebagai fungsi sigmoid. **(c)** Plot hipotesis regresi logistik $h_w(x) = Logistik(w \cdot x)$ untuk data yang ditunjukkan pada Gambar 9.15(b).

memiliki sifat matematika yang lebih praktis. Fungsi ini ditunjukkan pada Gambar 9.17(b). Dengan fungsi logistik menggantikan fungsi ambang batas, kita sekarang memiliki

$$h_w(x) = Logistik(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}.$$

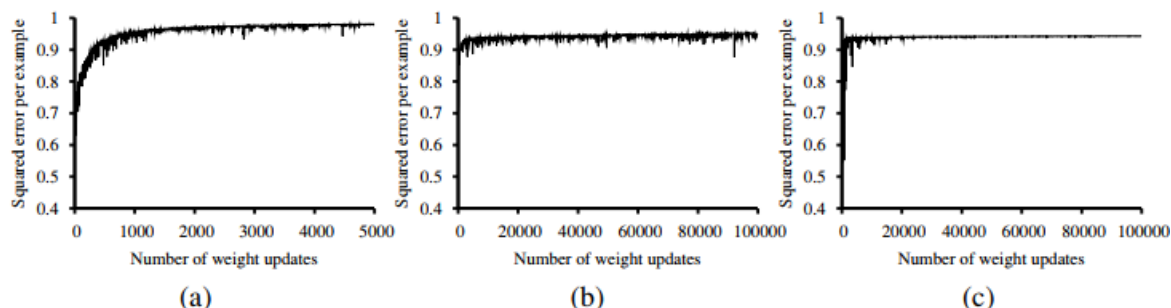
Contoh hipotesis semacam itu untuk masalah gempa bumi/ledakan dua masukan ditunjukkan pada Gambar 9.17(c). Perhatikan bahwa keluarannya, berupa angka antara 0 dan 1, dapat diinterpretasikan sebagai *probabilitas* untuk masuk ke dalam kelas berlabel 1. Hipotesis ini membentuk batas lunak dalam ruang masukan dan memberikan probabilitas 0,5 untuk setiap masukan di pusat wilayah batas, dan mendekati 0 atau 1 saat kita menjauh dari batas.

Proses penyesuaian bobot model ini untuk meminimalkan kerugian pada kumpulan data disebut **regresi logistik**. Tidak ada solusi bentuk tertutup yang mudah untuk menemukan nilai w optimal dengan model ini, tetapi perhitungan penurunan gradiennya mudah. Karena hipotesis kita tidak lagi hanya menghasilkan 0 atau 1, kita akan menggunakan fungsi kerugian L_2 ; selain itu, agar rumus tetap mudah dibaca, kita akan menggunakan g untuk mewakili fungsi logistik, dengan g' sebagai turunannya.

Untuk satu contoh (x, y) , turunan gradien sama dengan turunan regresi linier (Persamaan (9.5)) hingga ke titik di mana bentuk h yang sebenarnya dimasukkan. (Untuk turunan ini, kita memerlukan **aturan rantai**: $\partial g(f(x))/\partial x = g'(f(x)) \partial f(x)/\partial x$.) Kita memiliki

$$\begin{aligned} \frac{\partial}{\partial w_i} Loss(w) &= \frac{\partial}{\partial w_i} (y - h_w(x))^2 \\ &= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i} (y - h_w(x)) \end{aligned}$$

$$\begin{aligned}
&= -2(y - h_w(x)) \times g'(w \cdot x) \times \frac{\partial}{\partial w_i} w \cdot x \\
&= -2(y - h_w(x)) \times g'(w \cdot x) \times x_i.
\end{aligned}$$



Gambar 9.18 Pengulangan eksperimen pada Gambar 9.16 menggunakan regresi logistik dan galat kuadrat. Plot pada (a) mencakup 5000 iterasi, bukan 1000, sementara (b) dan (c) menggunakan skala yang sama.

Turunan g' dari fungsi logistik memenuhi $g'(z) = g(z)(1 - g(z))$, jadi kita punya

$$g'(w \cdot x) = g(w \cdot x)(1 - g(w \cdot x)) = h_w(x)(1 - h_w(x))$$

jadi pembaruan berat untuk meminimalkan kerugian adalah

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times h_w(x)(1 - h_w(x)) \times x_i. \quad (18.8)$$

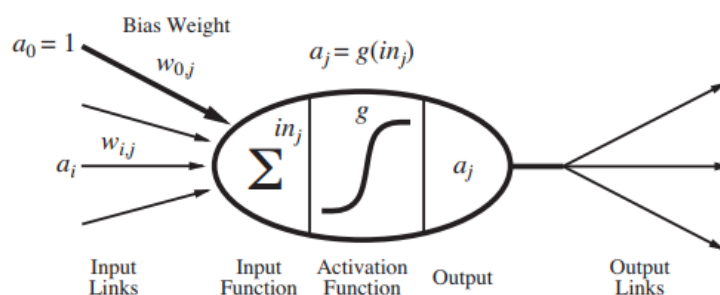
Dengan mengulangi eksperimen pada Gambar 9.16 dengan regresi logistik, alih-alih pengklasifikasi ambang batas linier, kami memperoleh hasil yang ditunjukkan pada Gambar 9.18. Pada (a), kasus yang dapat dipisahkan secara linier, regresi logistik agak lebih lambat konvergen, tetapi berperilaku jauh lebih terprediksi. Pada (b) dan (c), di mana datanya bising dan tidak dapat dipisahkan, regresi logistik konvergen jauh lebih cepat dan andal. Keunggulan ini cenderung terbawa ke dalam aplikasi dunia nyata dan regresi logistik telah menjadi salah satu teknik klasifikasi paling populer untuk permasalahan dalam kedokteran, analisis pemasaran dan survei, penilaian kredit, kesehatan masyarakat, dan aplikasi lainnya.

9.7 JARINGAN SYARAF TIRUAN

Sekarang kita beralih ke topik yang tampaknya agak tidak berhubungan: otak. Faktanya, seperti yang akan kita lihat, gagasan teknis yang telah kita bahas sejauh ini dalam bab ini ternyata bermanfaat dalam membangun model matematika aktivitas otak; sebaliknya, berpikir tentang otak telah membantu memperluas cakupan gagasan teknis tersebut.

Bab 1 menyentuh secara singkat temuan dasar ilmu saraf, khususnya hipotesis bahwa aktivitas mental terutama terdiri dari aktivitas elektrokimia dalam jaringan sel-sel otak yang disebut **neuron**. (Gambar 1.2 menunjukkan diagram skematis dari neuron pada umumnya.)

Terinspirasi oleh hipotesis ini, beberapa karya AI paling awal bertujuan untuk menciptakan **jaringan saraf tiruan**. (Nama lain untuk bidang ini termasuk **koneksionisme**, **pemrosesan terdistribusi paralel**, dan **komputasi saraf**.) Gambar 9.19 menunjukkan model matematika sederhana dari neuron yang dirancang oleh McCulloch dan Pitts (1943). Secara kasar, ia "menembak" ketika kombinasi linear dari inputnya melebihi ambang batas tertentu (baik keras maupun lunak), yang berarti ia mengimplementasikan pengklasifikasi linear seperti yang dijelaskan di bagian sebelumnya. Jaringan saraf hanyalah kumpulan unit-unit yang saling terhubung; properti jaringan ditentukan oleh topologinya dan properti dari "neuron" itu sendiri.



Gambar 9.19 Model matematika sederhana untuk sebuah neuron. Aktivasi keluaran unit adalah $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, di mana a_i adalah aktivasi keluaran unit i dan $w_{i,j}$ adalah bobot pada tautan dari unit i ke unit ini.

Sejak tahun 1943, model yang jauh lebih detail dan realistis telah dikembangkan, baik untuk neuron maupun sistem yang lebih besar di otak, yang mengarah pada bidang **ilmu saraf komputasional** modern. Di sisi lain, para peneliti di bidang AI dan statistika mulai tertarik pada sifat-sifat jaringan saraf yang lebih abstrak, seperti kemampuannya untuk melakukan komputasi terdistribusi, menoleransi masukan yang bising, dan belajar. Meskipun kita sekarang memahami bahwa jenis sistem lain—termasuk jaringan Bayesian—memiliki sifat-sifat ini, jaringan saraf tetap menjadi salah satu bentuk sistem pembelajaran yang paling populer dan efektif serta layak untuk dipelajari.

Struktur Jaringan Saraf Tiruan

Jaringan saraf tiruan terdiri dari simpul atau **unit** yang terhubung oleh **tautan** terarah. Tautan dari unit i ke unit j berfungsi untuk menyebarkan **aktivasi** a_i dari i ke j . Setiap tautan juga memiliki **bobot** numerik $w_{i,j}$ yang terkait dengannya, yang menentukan kekuatan dan tanda koneksi. Seperti halnya dalam model regresi linier, setiap unit memiliki masukan dummy $a_0 = 1$ dengan bobot terkait $w_{0,j}$. Setiap unit j pertama-tama menghitung jumlah terbobot dari masukannya:

$$in_j = \sum_{i=0}^n w_{i,j} a_i.$$

Kemudian ia menerapkan **fungsi aktivasi** g pada jumlah ini untuk memperoleh output:

$$a_j = g(in_j) = g\left(\sum_{i=0}^n w_{i,j} a_i\right). \quad (9.9)$$

Fungsi aktivasi g biasanya berupa ambang batas keras (Gambar 9.17(a)), yang dalam hal ini unitnya disebut **perceptron**, atau fungsi logistik (Gambar 9.17(b)), yang dalam hal ini istilah **perceptron sigmoid** terkadang digunakan. Kedua fungsi aktivasi nonlinier ini memastikan sifat penting bahwa seluruh jaringan unit dapat merepresentasikan fungsi nonlinier. Sebagaimana disebutkan dalam pembahasan regresi logistik, fungsi aktivasi logistik memiliki keuntungan tambahan karena dapat diturunkan.

Setelah menentukan model matematika untuk masing-masing "neuron", tugas selanjutnya adalah menghubungkan mereka bersama-sama untuk membentuk sebuah jaringan. Ada dua cara yang secara fundamental berbeda untuk melakukan hal ini. **Jaringan umpan-maju** memiliki koneksi hanya dalam satu arah—yaitu, membentuk grafik asiklik berarah. Setiap simpul menerima masukan dari simpul "hulu" dan mengirimkan keluaran ke simpul "hilir"; tidak ada loop. Jaringan umpan-maju merepresentasikan fungsi dari masukannya saat ini; Dengan demikian, ia tidak memiliki keadaan internal selain bobot itu sendiri. **Jaringan rekuren**, di sisi lain, memasukkan keluarannya kembali ke masukannya sendiri. Ini berarti bahwa tingkat aktivasi jaringan membentuk sistem dinamis yang dapat mencapai keadaan stabil atau menunjukkan osilasi atau bahkan perilaku kacau. Selain itu, respons jaringan terhadap masukan tertentu bergantung pada keadaan awalnya, yang mungkin bergantung pada masukan sebelumnya. Oleh karena itu, jaringan rekuren (tidak seperti jaringan umpan-maju) dapat mendukung memori jangka pendek. Hal ini membuatnya lebih menarik sebagai model otak, tetapi juga lebih sulit untuk dipahami. Bagian ini akan berfokus pada jaringan umpan-maju; beberapa petunjuk untuk bacaan lebih lanjut tentang jaringan rekuren diberikan di akhir bab ini.

Jaringan umpan-maju biasanya disusun dalam **lapisan**, sehingga setiap unit hanya menerima masukan dari unit-unit di lapisan sebelumnya. Dalam dua subbagian berikutnya, kita akan membahas jaringan lapis tunggal, di mana setiap unit terhubung langsung dari masukan jaringan ke keluarannya, dan jaringan lapis ganda, yang memiliki satu atau lebih lapisan **unit tersembunyi** yang tidak terhubung ke keluaran jaringan. Sejauh ini dalam bab ini, kita hanya membahas permasalahan pembelajaran dengan satu variabel keluaran y , tetapi jaringan saraf tiruan sering digunakan dalam kasus-kasus di mana beberapa keluaran diperlukan. Misalnya, jika kita ingin melatih jaringan untuk menambahkan dua bit masukan, masing-masing 0 atau 1, kita memerlukan satu keluaran untuk bit penjumlahan dan satu untuk bit pembawa. Selain itu, ketika permasalahan pembelajaran melibatkan klasifikasi ke dalam lebih dari dua kelas—misalnya, ketika mempelajari cara mengkategorikan gambar digit tulisan tangan—umumnya digunakan satu unit keluaran untuk setiap kelas.

Jaringan saraf tiruan umpan-maju lapis tunggal (perceptron)

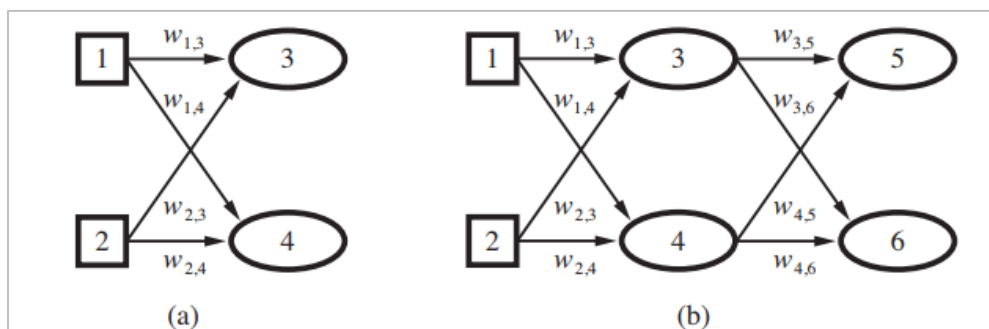
Jaringan dengan semua masukan terhubung langsung ke keluaran disebut **jaringan saraf tiruan lapis tunggal**, atau **jaringan perceptron**. Gambar 9.20 menunjukkan jaringan perceptron sederhana dengan dua masukan dan dua keluaran. Dengan jaringan seperti itu, kita mungkin ingin mempelajari fungsi penjumlah dua bit, misalnya. Berikut semua data pelatihan yang kita perlukan:

x_1	x_2	$y_3(\text{membawa})$	$y_4(\text{jumlah})$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

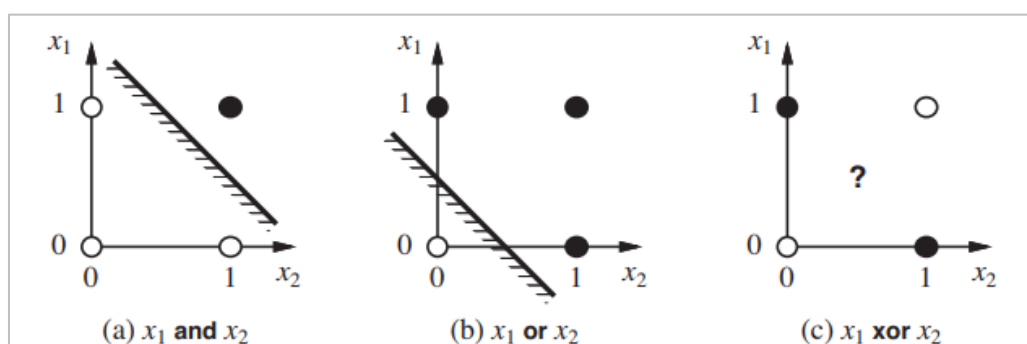
Hal pertama yang perlu diperhatikan adalah bahwa jaringan perceptron dengan m keluaran sebenarnya merupakan m jaringan terpisah, karena setiap bobot hanya memengaruhi satu keluaran. Dengan demikian, akan ada m proses pelatihan terpisah. Lebih lanjut, bergantung pada jenis fungsi aktivasi yang digunakan, proses pelatihan akan berupa **aturan pembelajaran perceptron** (Persamaan (9.7)) atau aturan penurunan gradien untuk **regresi logistik** (Persamaan (9.8)).

Jika Anda mencoba salah satu metode pada data penjumlah dua-bit, sesuatu yang menarik akan terjadi. Unit 3 mempelajari fungsi carry dengan mudah, tetapi unit 4 sama sekali gagal mempelajari fungsi penjumlahan. Tidak, unit 4 tidak cacat! Masalahnya ada pada fungsi penjumlahan itu sendiri. Kita telah melihat di Bagian 18.6 bahwa pengklasifikasi linier (baik keras maupun lunak) dapat merepresentasikan batas keputusan linier dalam ruang masukan. Ini berfungsi dengan baik untuk fungsi carry, yang merupakan logika AND (lihat Gambar 9.21(a)). Fungsi penjumlahan, bagaimanapun, adalah XOR (eksklusif OR) dari dua masukan. Seperti yang diilustrasikan Gambar 9.21(c), fungsi ini tidak dapat dipisahkan secara linear sehingga perceptron tidak dapat mempelajarinya.

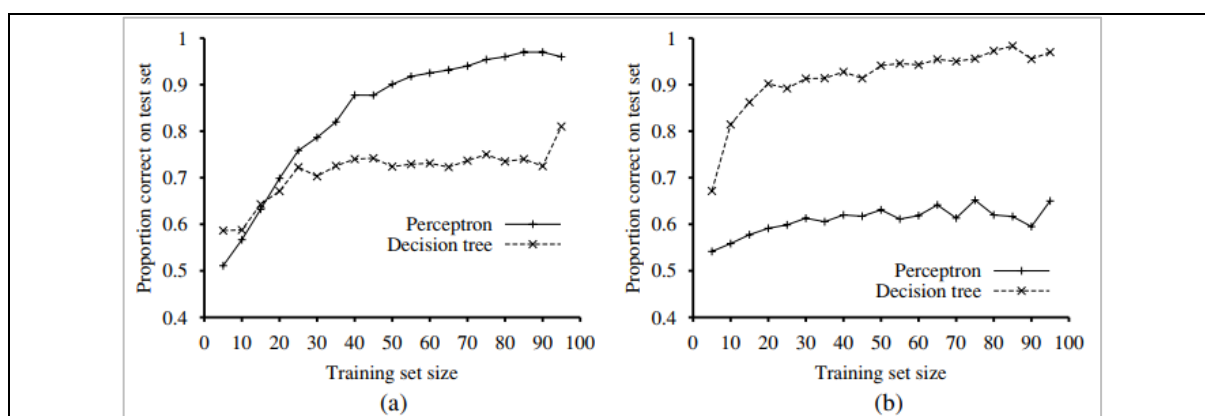
Fungsi yang dapat dipisahkan secara linear hanya merupakan sebagian kecil dari semua fungsi Boolean. Ketidakmampuan perceptron untuk mempelajari bahkan fungsi sederhana seperti XOR merupakan kemunduran yang signifikan bagi komunitas jaringan saraf yang baru lahir pada tahun 1960-an. Namun, perceptron jauh dari tidak berguna. Bagian 9.6 mencatat bahwa regresi logistik (yaitu, melatih perceptron sigmoid) bahkan saat ini merupakan alat yang sangat populer dan efektif. Selain itu, sebuah perceptron dapat merepresentasikan beberapa fungsi Boolean yang cukup "kompleks" dengan sangat ringkas. Misalnya, **fungsi mayoritas**, yang menghasilkan 1 hanya jika lebih dari separuh n masukannya bernilai 1, dapat direpresentasikan oleh sebuah perceptron dengan setiap $w_i = 1$ dan dengan $w_0 = -n/2$. Sebuah pohon keputusan membutuhkan jumlah simpul yang eksponensial untuk merepresentasikan fungsi ini.



Gambar 9.20 (a) Jaringan perseptron dengan dua masukan dan dua unit keluaran. (b) Jaringan saraf tiruan dengan dua masukan, satu lapisan tersembunyi dengan dua unit, dan satu unit keluaran. Masukan dummy dan bobot terkait tidak ditampilkan.



Gambar 9.21 Keterpisahan linear dalam perseptron ambang batas. Titik hitam menunjukkan titik di ruang masukan dengan nilai fungsi 1, dan titik putih menunjukkan titik dengan nilai 0. Perseptron mengembalikan nilai 1 pada wilayah di sisi garis yang tidak diarsir. Pada (c), tidak ada garis yang mengklasifikasikan masukan dengan benar.



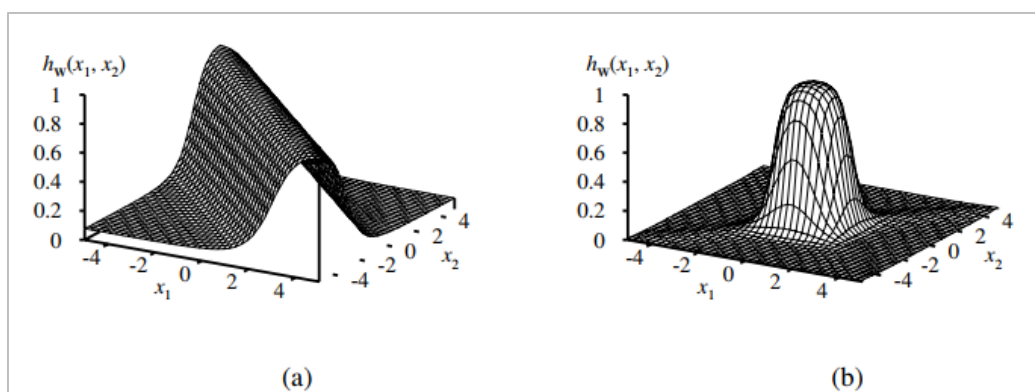
Gambar 9.22 Membandingkan kinerja persepsi dan pohon keputusan. (a) Persepsi lebih baik dalam mempelajari fungsi mayoritas dari 11 masukan. (b) Pohon keputusan lebih baik dalam mempelajari predikat *WillWait* dalam contoh restoran.

Gambar 9.22 menunjukkan kurva pembelajaran untuk sebuah perceptron pada dua permasalahan yang berbeda. Di sebelah kiri, kami menunjukkan kurva untuk mempelajari fungsi mayoritas dengan 11 masukan Boolean (yaitu, fungsi menghasilkan 1 jika 6 atau lebih

masuk bernilai 1). Seperti yang kami perkirakan, perceptron mempelajari fungsi tersebut dengan cukup cepat, karena fungsi mayoritas dapat dipisahkan secara linear. Di sisi lain, pembelajar pohon keputusan tidak mengalami kemajuan, karena fungsi mayoritas sangat sulit (meskipun bukan tidak mungkin) untuk direpresentasikan sebagai pohon keputusan. Di sebelah kanan, kami memiliki contoh restoran. Permasalahan solusi mudah direpresentasikan sebagai pohon keputusan, tetapi tidak dapat dipisahkan secara linear. Bidang terbaik melalui data hanya mengklasifikasikan dengan benar 65%.

Jaringan saraf tiruan umpan-maju multilapis

Satu unit ambang batas tidak akan menyelesaikan semua masalah mereka. Faktanya, makalah mereka membuktikan bahwa unit tersebut dapat merepresentasikan fungsi Boolean dasar AND, OR, dan NOT, dan kemudian berargumen bahwa fungsionalitas apa pun yang diinginkan dapat diperoleh dengan menghubungkan sejumlah besar unit ke dalam jaringan (yang mungkin berulang) dengan kedalaman yang berubah-ubah. Masalahnya adalah tidak ada yang tahu cara melatih jaringan semacam itu.



Gambar 9.23 (a) Hasil penggabungan dua fungsi ambang batas lunak yang saling berhadapan untuk menghasilkan sebuah punggungan. (b) Hasil penggabungan dua punggungan untuk menghasilkan sebuah tonjolan.

Hal ini ternyata menjadi masalah yang mudah jika kita menganggap jaringan dengan cara yang benar: sebagai fungsi $h_w(x)$ yang diparameterisasi oleh bobot w . Perhatikan jaringan sederhana yang ditunjukkan pada Gambar 9.20(b), yang memiliki dua unit masukan, dua unit tersembunyi, dan dua unit keluaran. (Selain itu, setiap unit memiliki input dummy yang ditetapkan pada 1.) Dengan vektor input $x = (x_1, x_2)$, aktivasi unit input ditetapkan menjadi $(a_1, a_2) = (x_1, x_2)$. Output pada unit 5 diberikan oleh

$$\begin{aligned} a_5 &= g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\ &= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5}g(w_{0,4} + w_{1,4}a_1 + w_{2,4}a_2)) \\ &= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)). \end{aligned}$$

Dengan demikian, keluaran dinyatakan sebagai fungsi masukan dan bobot. Ekspresi serupa berlaku untuk unit 6. Selama kita dapat menghitung turunan dari ekspresi tersebut terhadap bobot, kita dapat menggunakan metode minimisasi kerugian penurunan gradien untuk melatih jaringan. Bagian 9.7 menunjukkan cara melakukannya secara tepat. Dan karena fungsi yang direpresentasikan oleh jaringan dapat sangat nonlinier—terdiri dari fungsi ambang batas lunak nonlinier bersarang—kita dapat melihat jaringan saraf tiruan sebagai alat untuk melakukan **regresi nonlinier**.

Sebelum mempelajari aturan pembelajaran, mari kita lihat cara jaringan menghasilkan fungsi yang rumit. Pertama, ingatlah bahwa setiap unit dalam jaringan sigmoid merepresentasikan ambang batas lunak dalam ruang masukannya, seperti yang ditunjukkan pada Gambar 9.17(c). Dengan satu lapisan tersembunyi dan satu lapisan keluaran, seperti pada Gambar 9.20(b), setiap unit keluaran menghitung kombinasi linier ambang batas lunak dari beberapa fungsi tersebut. Misalnya, dengan menambahkan dua fungsi ambang batas lunak yang saling berhadapan dan melakukan ambang batas terhadap hasilnya, kita dapat memperoleh fungsi "ridge" seperti yang ditunjukkan pada Gambar 9.23(a). Dengan menggabungkan dua ridge tersebut pada sudut siku-siku satu sama lain (yaitu, menggabungkan keluaran dari empat unit tersembunyi), kita memperoleh "bump" seperti yang ditunjukkan pada Gambar 9.23(b).

Dengan lebih banyak unit tersembunyi, kita dapat menghasilkan lebih banyak bump dengan ukuran berbeda di lebih banyak tempat. Bahkan, dengan satu lapisan tersembunyi yang cukup besar, dimungkinkan untuk merepresentasikan fungsi kontinu apa pun dari masukan dengan akurasi yang sewenang-wenang; dengan dua lapisan, bahkan fungsi diskontinu pun dapat direpresentasikan.⁹ Sayangnya, untuk struktur jaringan *tertentu*, lebih sulit untuk mengkarakterisasi secara tepat fungsi mana yang dapat direpresentasikan dan mana yang tidak.

Pembelajaran dalam jaringan multilapis

Pertama, mari kita singkirkan satu komplikasi kecil yang muncul dalam jaringan multilapis: interaksi antar masalah pembelajaran ketika jaringan memiliki beberapa keluaran. Dalam kasus seperti itu, kita harus menganggap jaringan tersebut mengimplementasikan fungsi vektor $\mathbf{h}_w$, alih-alih fungsi skalar h_w ; misalnya, jaringan pada Gambar 9.20(b) menghasilkan vektor $[a_5, a_6]$. Demikian pula, keluaran target akan berupa vektor $\mathbf{y}$. Sementara jaringan perceptron terurai menjadi m masalah pembelajaran terpisah untuk masalah m -keluaran, penguraian ini gagal dalam jaringan multilapis. Misalnya, baik a_5 maupun a_6 pada Gambar 9.20(b) bergantung pada semua bobot lapisan masukan, sehingga pembaruan bobot tersebut akan bergantung pada galat pada a_5 maupun a_6 . Untungnya, ketergantungan ini sangat sederhana dalam kasus fungsi kerugian apa pun yang bersifat *aditif* di seluruh komponen vektor galat $\mathbf{y} - \mathbf{h}_w(\mathbf{x})$. Untuk kerugian L_2 , kita peroleh, untuk bobot apa pun w ,

$$\frac{\partial}{\partial \mathbf{w}} \text{Loss}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} |\mathbf{y} - \mathbf{h}_w(\mathbf{x})|^2 = \frac{\partial}{\partial \mathbf{w}} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial \mathbf{w}} (y_k - a_k)^2 \quad (9.10)$$

di mana indeks k berkisar di atas simpul-simpul pada lapisan keluaran. Setiap suku dalam penjumlahan akhir hanyalah gradien kerugian untuk keluaran ke- k , yang dihitung seolah-olah keluaran lainnya tidak ada. Oleh karena itu, kita dapat menguraikan masalah pembelajaran keluaran- m menjadi masalah pembelajaran m , asalkan kita ingat untuk menjumlahkan kontribusi gradien dari masing-masing masalah saat memperbarui bobot.

Komplikasi utama berasal dari penambahan lapisan tersembunyi ke jaringan. Meskipun kesalahan $\mathbf{y} - \mathbf{h}_w$ pada lapisan keluaran jelas, kesalahan pada lapisan tersembunyi tampak misterius karena data pelatihan tidak menunjukkan nilai yang seharusnya dimiliki simpul-simpul tersembunyi. Untungnya, ternyata kita dapat melakukan **propagasi balik** kesalahan dari lapisan keluaran ke lapisan tersembunyi. Proses propagasi balik muncul langsung dari derivasi gradien kesalahan keseluruhan. Pertama, kita akan menjelaskan proses tersebut dengan justifikasi intuitif; kemudian, kita akan menunjukkan derivasinya.

Pada lapisan keluaran, aturan pembaruan bobot identik dengan Persamaan (9.8). Kita memiliki beberapa unit keluaran, jadi misalkan Err_k adalah komponen ke- k dari vektor kesalahan $\mathbf{y} - \mathbf{h}_w$. Kita juga akan merasa lebih mudah untuk mendefinisikan kesalahan termodifikasi $\Delta_k = Err_k \times g'(in_k)$, sehingga aturan pembaruan bobot menjadi

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k. \quad (9.11)$$

Untuk memperbarui koneksi antara unit masukan dan unit tersembunyi, kita perlu mendefinisikan kuantitas yang analog dengan suku kesalahan untuk simpul keluaran. Di sinilah kita melakukan propagasi balik kesalahan. Idennya adalah bahwa simpul tersembunyi j "bertanggung jawab" atas sebagian kesalahan Δ_k di setiap simpul keluaran yang terhubung dengannya. Dengan demikian, nilai Δ_k dibagi berdasarkan kekuatan koneksi antara simpul tersembunyi dan simpul keluaran dan dipropagasi balik untuk menghasilkan nilai Δ_j untuk lapisan tersembunyi. Aturan propagasi untuk nilai Δ adalah sebagai berikut:

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k. \quad (9.12)$$

```

function BACK – PROP – LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
           network, a multilayer network with L layers, weights  $w_{i,j}$ , activation function g
  local variables:  $\Delta$ , a vector of errors, indexed by network node

  repeat
    for each weight  $w_{i,j}$  in network do
       $w_{i,j} \leftarrow$  a small random number
    for each example (x, y) in examples do
      /* Propagate the inputs forward to compute the outputs */
      for each node i in the input layer do
         $a_i \leftarrow x_i$ 
      for  $l = 2$  to L do
        for each node j in layer l do
           $in_j \leftarrow \sum_i w_{i,j} a_i$ 
           $a_j \leftarrow g(in_j)$ 
      /* Propagate deltas backward from output layer to input layer */
      for each node j in the output layer do
         $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$ 
      for  $l = L - 1$  to 1 do
        for each node i in layer l do
           $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 
      /* Update every weight in network using deltas */
      for each weight  $w_{i,j}$  in network do
         $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ 
  until some stopping criterion is satisfied
  return network

```

Gambar 9.24 Algoritma back-propagation untuk pembelajaran dalam jaringan multilayer.

Sekarang aturan pembaruan bobot untuk bobot antara masukan dan lapisan tersembunyi pada dasarnya identik dengan aturan pembaruan untuk lapisan keluaran:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j.$$

Proses back-propagation dapat diringkas sebagai berikut:

- Hitung nilai Δ untuk unit keluaran, menggunakan galat yang teramati.
- Dimulai dengan lapisan keluaran, ulangi langkah-langkah berikut untuk setiap lapisan dalam jaringan, hingga lapisan tersembunyi paling awal tercapai:
 - Propagasi nilai Δ kembali ke lapisan sebelumnya.
 - Perbarui bobot antara kedua lapisan.

Algoritme detailnya ditunjukkan pada Gambar 9.24.

Bagi yang lebih menyukai matematika, kita sekarang akan menurunkan persamaan back-propagation dari prinsip-prinsip dasar. Penurunannya cukup mirip dengan perhitungan gradien untuk regresi logistik (mengarah ke Persamaan (9.8)), kecuali bahwa kita harus menggunakan aturan rantai lebih dari sekali.

Mengikuti Persamaan (9.10), kita hanya menghitung gradien untuk $Loss_k = (y_k - a_k)^2$ pada keluaran ke- k . Gradien kerugian ini terhadap bobot yang menghubungkan lapisan tersembunyi ke lapisan keluaran akan menjadi nol, kecuali untuk bobot $w_{j,k}$ yang terhubung ke unit keluaran ke- k . Untuk bobot tersebut, kita memiliki

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{j,k}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{j,k}} \\ &= -2(y_k - a_k) g'(in_k) \frac{\partial in_k}{\partial w_{j,k}} = -2(y_k - a_k) g'(in_k) \frac{\partial}{\partial w_{j,k}} \left(\sum_j w_{j,k} a_j \right) \\ &= -2(y_k - a_k) g'(in_k) a_j = -a_j \Delta_k,\end{aligned}$$

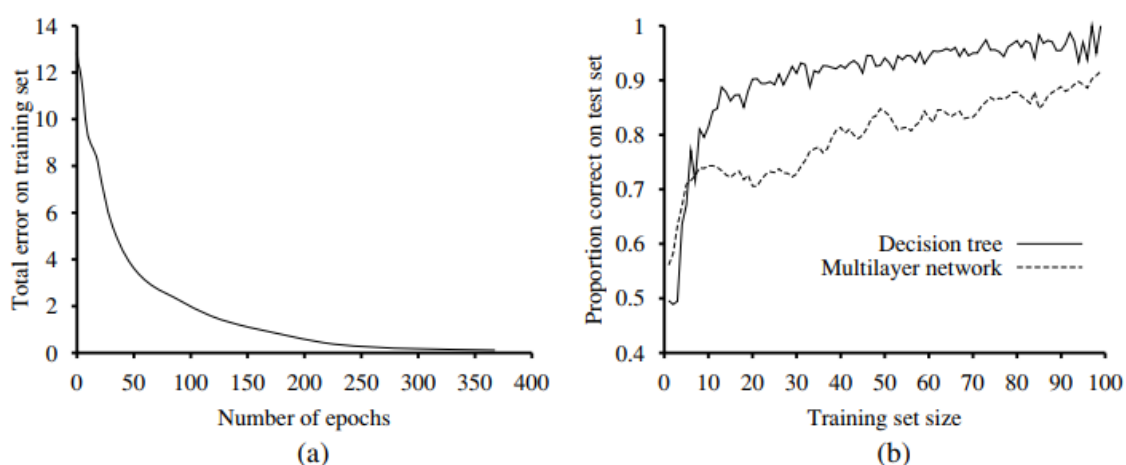
dengan Δ_k yang didefinisikan seperti sebelumnya. Untuk mendapatkan gradien terhadap bobot $w_{i,j}$ yang menghubungkan lapisan masukan ke lapisan tersembunyi, kita harus mengekspansi aktivasi a_j dan menerapkan kembali aturan rantai. Kita akan menunjukkan derivasinya secara detail karena menarik untuk melihat bagaimana operator turunan merambat kembali melalui jaringan:

$$\begin{aligned}\frac{\partial Loss_k}{\partial w_{i,j}} &= -2(y_k - a_k) \frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k) \frac{\partial g(in_k)}{\partial w_{i,j}} \\ &= -2(y_k - a_k) g'(in_k) \frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k \frac{\partial}{\partial w_{i,j}} \left(\sum_j w_{j,k} a_j \right) \\ &= -2\Delta_k w_{j,k} \frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k} \frac{\partial g(in_j)}{\partial w_{i,j}} \\ &= -2\Delta_k w_{j,k} g'(in_j) \frac{\partial in_j}{\partial w_{i,j}} \\ &= -2\Delta_k w_{j,k} g'(in_j) \frac{\partial}{\partial w_{i,j}} \left(\sum_i w_{i,j} a_i \right) \\ &= -2\Delta_k w_{j,k} g'(in_j) a_i = -a_i \Delta_j,\end{aligned}$$

di mana Δ_j didefinisikan seperti sebelumnya. Dengan demikian, kita memperoleh aturan pembaruan yang diperoleh sebelumnya dari pertimbangan intuitif. Jelas juga bahwa proses ini dapat dilanjutkan untuk jaringan dengan lebih dari satu lapisan tersembunyi, yang membenarkan algoritma umum yang diberikan pada Gambar 9.24.

Setelah melewati (atau melewati) semua matematika, mari kita lihat bagaimana jaringan satu lapisan tersembunyi bekerja pada masalah restoran. Pertama, kita perlu menentukan struktur jaringan. Kita memiliki 10 atribut yang menjelaskan setiap contoh, jadi kita membutuhkan 10 unit masukan. Haruskah kita memiliki satu lapisan tersembunyi atau dua? Berapa banyak simpul di setiap lapisan? Haruskah mereka terhubung sepenuhnya? Tidak

ada teori yang baik yang akan memberi tahu kita jawabannya. (Lihat bagian selanjutnya.) Seperti biasa, kita dapat menggunakan validasi silang: coba beberapa struktur berbeda dan lihat mana yang paling berhasil. Ternyata jaringan dengan satu lapisan tersembunyi yang berisi empat simpul kurang lebih tepat untuk masalah ini. Pada Gambar 9.25, kita menunjukkan dua kurva. Yang pertama adalah kurva pelatihan yang menunjukkan rata-rata kuadrat galat pada set pelatihan 100 contoh restoran selama proses pembaruan bobot. Hal ini menunjukkan bahwa jaringan memang konvergen hingga mencapai kesesuaian sempurna dengan data pelatihan. Kurva kedua adalah kurva pembelajaran standar untuk data restoran. Jaringan saraf tiruan memang belajar dengan baik, meskipun tidak secepat pembelajaran pohon keputusan; hal ini mungkin tidak mengejutkan, karena data tersebut awalnya dihasilkan dari pohon keputusan sederhana.



Gambar 9.25 (a) Kurva pelatihan yang menunjukkan pengurangan kesalahan secara bertahap seiring dengan modifikasi bobot selama beberapa periode, untuk serangkaian contoh tertentu dalam domain restoran. (b) Kurva pembelajaran komparatif yang menunjukkan bahwa pembelajaran pohon keputusan sedikit lebih baik dalam menyelesaikan masalah restoran dibandingkan dengan back-propagation dalam jaringan berlapis-lapis.

Tentu saja, jaringan saraf tiruan mampu melakukan tugas pembelajaran yang jauh lebih kompleks, meskipun harus dikatakan bahwa diperlukan sedikit penyesuaian untuk mendapatkan struktur jaringan yang tepat dan mencapai konvergensi mendekati optimum global dalam ruang bobot. Terdapat puluhan ribu aplikasi jaringan saraf tiruan yang telah dipublikasikan.

Mempelajari Struktur Jaringan Saraf Tiruan

Sejauh ini, kita telah membahas masalah pembelajaran bobot, dengan struktur jaringan yang tetap; sama seperti jaringan Bayesian, kita juga perlu memahami cara menemukan struktur jaringan terbaik. Jika kita memilih jaringan yang terlalu besar, jaringan tersebut akan mampu mengingat semua contoh dengan membentuk tabel pencarian yang besar, tetapi belum tentu dapat digeneralisasi dengan baik ke input yang belum pernah dilihat

sebelumnya. Dengan kata lain, seperti semua model statistik, jaringan saraf tiruan rentan terhadap **overfitting** ketika terdapat terlalu banyak parameter dalam model. Kita melihat hal ini pada Gambar 9.1, di mana model berparameter tinggi pada (b) dan (c) dapat menyesuaikan semua data, tetapi mungkin tidak dapat digeneralisasi sebaik model berparameter rendah pada (a) dan (d).

Jika kita tetap menggunakan jaringan yang terhubung penuh, satu-satunya pilihan yang dapat dibuat adalah jumlah lapisan tersembunyi dan ukurannya. Pendekatan yang umum adalah mencoba beberapa lapisan dan mempertahankan yang terbaik. Teknik **validasi silang** pada Bab 18 diperlukan jika kita ingin menghindari **mengintip** set pengujian. Artinya, kita memilih arsitektur jaringan yang memberikan akurasi prediksi tertinggi pada set validasi.

Jika kita ingin mempertimbangkan jaringan yang tidak sepenuhnya terhubung, maka kita perlu menemukan beberapa metode pencarian yang efektif melalui ruang kemungkinan topologi koneksi yang sangat besar. Algoritma **kerusakan otak optimal** dimulai dengan jaringan yang sepenuhnya terhubung dan menghilangkan koneksi darinya. Setelah jaringan dilatih untuk pertama kalinya, pendekatan teoretis informasi mengidentifikasi pilihan koneksi optimal yang dapat dihilangkan. Jaringan kemudian dilatih ulang, dan jika kinerjanya tidak menurun, proses diulang. Selain menghilangkan koneksi, juga dimungkinkan untuk menghilangkan unit yang tidak berkontribusi banyak pada hasil.

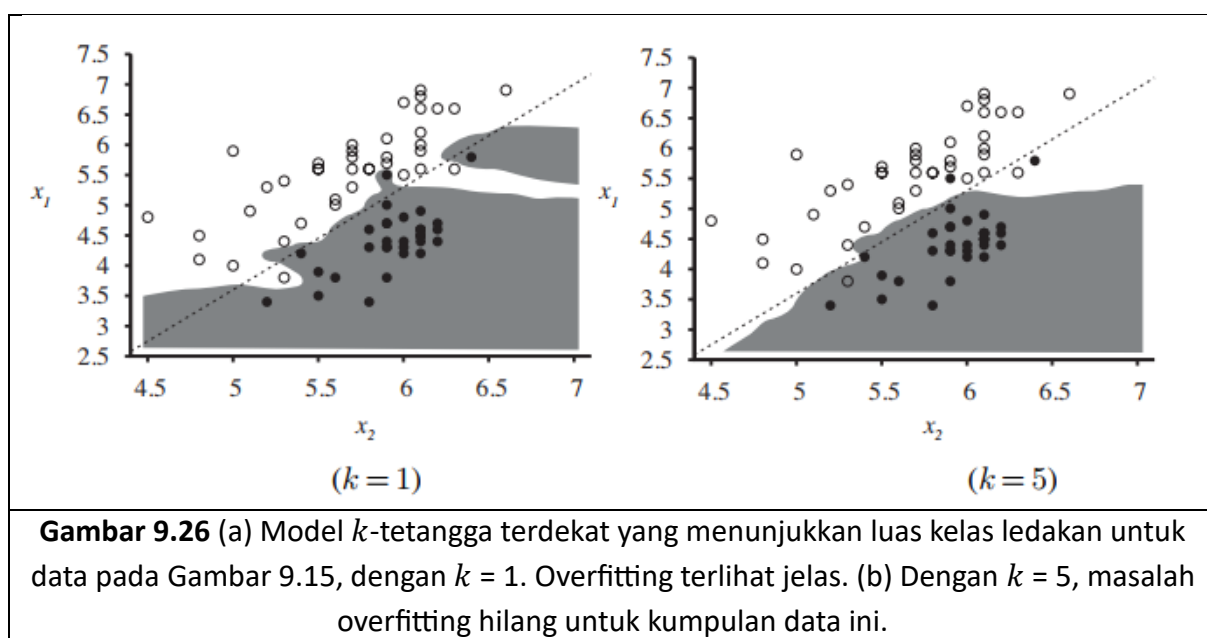
Beberapa algoritma telah diusulkan untuk mengembangkan jaringan yang lebih besar dari jaringan yang lebih kecil. Salah satunya, algoritma **tiling**, menyerupai pembelajaran daftar keputusan. Idennya adalah memulai dengan satu unit yang berusaha sebaik mungkin untuk menghasilkan keluaran yang benar pada sebanyak mungkin contoh pelatihan. Unit-unit berikutnya ditambahkan untuk menangani contoh-contoh yang salah pada unit pertama. Algoritme hanya menambahkan unit sebanyak yang dibutuhkan untuk mencakup semua contoh.

9.8 MODEL NONPARAMETRIK

Regresi linear dan jaringan saraf tiruan menggunakan data latih untuk mengestimasi serangkaian parameter tetap $\mathbf{w}$. Hal ini mendefinisikan hipotesis kita $h_{\mathbf{w}}(\mathbf{x})$, dan pada titik tersebut kita dapat membuang data latih, karena semuanya diringkas oleh $\mathbf{w}$. Model pembelajaran yang meringkas data dengan serangkaian parameter berukuran tetap (terlepas dari jumlah contoh pelatihan) disebut **model parametrik**.

Sebanyak apa pun data yang Anda masukkan ke dalam model parametrik, model tersebut tidak akan berubah pikiran tentang berapa banyak parameter yang dibutuhkannya. Ketika kumpulan data kecil, masuk akal untuk memiliki batasan yang kuat pada hipotesis yang diizinkan, untuk menghindari overfitting. Namun ketika ada ribuan atau jutaan atau miliaran contoh untuk dipelajari, tampaknya lebih baik membiarkan data berbicara sendiri daripada memaksanya berbicara melalui vektor parameter yang sangat kecil. Jika data menunjukkan bahwa jawaban yang benar adalah fungsi yang sangat bergelombang, kita tidak boleh membatasi diri pada fungsi linear atau fungsi yang sedikit bergelombang.

Model nonparametrik adalah model yang tidak dapat dikarakterisasi oleh serangkaian parameter yang dibatasi. Misalnya, anggaplah bahwa setiap hipotesis yang kita hasilkan hanya mempertahankan dalam dirinya sendiri semua contoh pelatihan dan menggunakan semuanya untuk memprediksi contoh berikutnya. Keluarga hipotesis seperti itu akan menjadi nonparametrik karena jumlah parameter efektif tidak terbatas—jumlahnya bertambah seiring dengan jumlah contoh. Pendekatan ini disebut **pembelajaran berbasis instan** atau **pembelajaran berbasis memori**. Metode pembelajaran berbasis instan yang paling sederhana adalah **pencarian tabel**: ambil semua contoh pelatihan, letakkan di tabel pencarian, lalu ketika diminta $h(\mathbf{x})$, lihat apakah $\mathbf{x}$ ada di tabel; jika ada, kembalikan y yang sesuai. Masalah dengan metode ini adalah tidak menggeneralisasi dengan baik: ketika $\mathbf{x}$ tidak ada di tabel, yang bisa dilakukannya hanyalah mengembalikan beberapa nilai default.



Model Tetangga Terdekat

Kita dapat menyempurnakan pencarian tabel dengan sedikit variasi: dengan kueri $\mathbf{x}_q$, temukan k contoh yang paling *dekat* dengan $\mathbf{x}_q$. Ini disebut pencarian **k -tetangga terdekat**. Kita akan menggunakan notasi $NN(k, \mathbf{x}_q)$ untuk menyatakan himpunan k tetangga terdekat.

Untuk melakukan klasifikasi, pertama-tama temukan $NN(k, \mathbf{x}_q)$, lalu ambil suara terbanyak dari tetangga (yang merupakan suara mayoritas dalam kasus klasifikasi biner). Untuk menghindari seri, k selalu dipilih sebagai angka ganjil. Untuk melakukan regresi, kita dapat mengambil rata-rata atau median dari k tetangga, atau kita dapat menyelesaikan masalah regresi linier pada tetangga.

Pada Gambar 9.26, kami menunjukkan batas keputusan klasifikasi k -tetangga terdekat untuk $k = 1$ dan 5 pada set data gempa bumi dari Gambar 9.15. Metode nonparametrik masih rentan terhadap underfitting dan overfitting, sama seperti metode parametrik. Dalam kasus ini, 1-tetangga terdekat merupakan overfitting; ia bereaksi terlalu berlebihan terhadap outlier hitam di kanan atas dan outlier putih di (5.4, 3.7). Batas keputusan 5-tetangga terdekat sudah

baik; k yang lebih tinggi akan menghasilkan underfitting. Seperti biasa, validasi silang dapat digunakan untuk memilih nilai k terbaik.

Kata "terdekat" sendiri menyiratkan metrik jarak. Bagaimana kita mengukur jarak dari titik kueri $\mathbf{x}_q$ ke titik contoh $\mathbf{x}_j$? Biasanya, jarak diukur dengan **jarak Minkowski** atau norma L^p , yang didefinisikan sebagai

$$L^p(x_j, x_q) = \left(\sum_i |x_{j,i} - x_{q,i}|^p \right)^{1/p}.$$

Dengan $p = 2$ ini adalah jarak Euclidean dan dengan $p = 1$ itu adalah jarak Manhattan. Dengan nilai atribut Boolean, jumlah atribut di mana dua titik berbeda disebut **jarak Hamming**. Seringkali $p = 2$ digunakan jika dimensi mengukur properti yang sama, seperti lebar, tinggi dan kedalaman bagian pada sabuk konveyor, dan jarak Manhattan digunakan jika tidak sama, seperti usia, berat, dan jenis kelamin pasien. Perhatikan bahwa jika kita menggunakan angka mentah dari setiap dimensi maka jarak total akan dipengaruhi oleh perubahan skala dalam dimensi apa pun. Artinya, jika kita mengubah dimensi i dari pengukuran dalam sentimeter ke mil sambil menjaga dimensi lain tetap sama, kita akan mendapatkan tetangga terdekat yang berbeda. Untuk menghindari hal ini, adalah umum untuk menerapkan **normalisasi** pada pengukuran di setiap dimensi. Satu pendekatan sederhana adalah menghitung mean μ_i dan deviasi standar σ_i dari nilai-nilai di setiap dimensi, dan mengubah skalanya sehingga $x_{j,i}$ menjadi $(x_{j,i} - \mu_i)/\sigma_i$. Metrik yang lebih kompleks, yang dikenal sebagai **jarak Mahalanobis**, memperhitungkan kovariansi antardimensi.

Dalam ruang berdimensi rendah dengan banyak data, tetangga terdekat bekerja dengan sangat baik: kita cenderung memiliki cukup titik data terdekat untuk mendapatkan jawaban yang baik. Namun, seiring bertambahnya jumlah dimensi, kita menghadapi masalah: tetangga terdekat dalam ruang berdimensi tinggi biasanya tidak terlalu dekat! Pertimbangkan k -tetangga terdekat pada himpunan data N titik yang terdistribusi secara seragam di seluruh interior hiperkubus satuan n -dimensi. Kita akan mendefinisikan lingkungan k -titik sebagai hiperkubus terkecil yang memuat k -tetangga terdekat. Misalkan ℓ adalah panjang sisi rata-rata suatu lingkungan. Maka volume lingkungan (yang memuat k titik) adalah ℓ^n dan volume kubus penuh (yang memuat N titik) adalah 1. Jadi, rata-rata, $\ell^n = k/N$. Dengan mengambil akar ke- n dari kedua sisi, kita memperoleh $\ell^n = (k/N)^{1/n}$.

Untuk konkretnya, misalkan $k = 10$ dan $N = 1.000.000$. Dalam dua dimensi ($n = 2$; sebuah persegi satuan), rata-rata lingkungan memiliki $\ell = 0,003$, sebagian kecil dari persegi satuan, dan dalam 3 dimensi, ℓ hanya 2% dari panjang rusuk kubus satuan. Namun, ketika kita mencapai 17 dimensi, ℓ adalah setengah dari panjang rusuk hiperkubus satuan, dan dalam 200 dimensi, ℓ adalah 94%. Masalah ini disebut **kutukan dimensionalitas**.

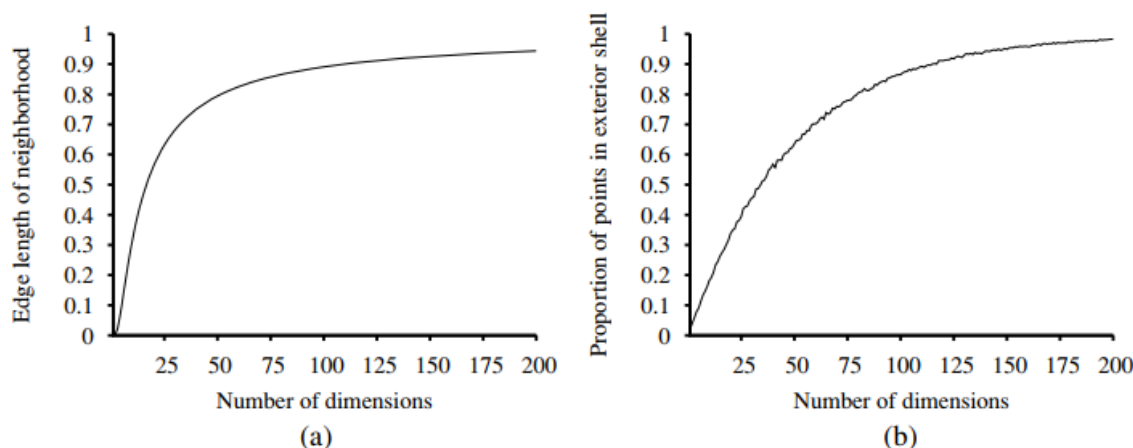
Cara lain untuk melihatnya: perhatikan titik-titik yang berada di dalam kulit tipis yang membentuk 1% terluar hiperkubus satuan. Ini adalah outlier; secara umum akan sulit untuk menemukan nilai yang tepat untuk mereka karena kita akan mengekstrapolasi alih-alih

menginterpolasi. Dalam satu dimensi, outlier ini hanya 2% dari titik-titik pada garis satuan (titik-titik di mana $x < .01$ atau $x > .99$), tetapi dalam 200 dimensi, lebih dari 98% titik berada dalam cangkang tipis ini—hampir semua titik adalah outlier. Anda dapat melihat contoh kecocokan tetangga terdekat yang buruk pada outlier jika Anda melihat Gambar 9.28(b).

Fungsi $NN(k, \mathbf{x}_q)$ secara konseptual mudah dipahami: diberikan satu set N contoh dan kueri $\mathbf{x}_q$, ulangi melalui contoh-contoh tersebut, ukur jarak ke $\mathbf{x}_q$ dari masing-masing contoh, dan pertahankan k terbaik. Jika kita puas dengan implementasi yang membutuhkan waktu eksekusi $O(N)$, maka itu adalah akhir cerita. Namun, metode berbasis instans dirancang untuk set data besar, jadi kita menginginkan algoritma dengan waktu proses sublinier. Analisis dasar algoritma menunjukkan bahwa pencarian tabel eksak adalah $O(N)$ dengan tabel sekuensial, $O(\log N)$ dengan pohon biner, dan $O(1)$ dengan tabel hash. Kita sekarang akan melihat bahwa pohon biner dan tabel hash juga dapat digunakan untuk menemukan tetangga terdekat.

Menemukan tetangga terdekat dengan pohon k-d

Pohon biner seimbang atas data dengan jumlah dimensi yang sembarang disebut **pohon k-d**, atau pohon k-dimensi. (Dalam notasi kami, jumlah dimensi adalah n , jadi mereka akan menjadi pohon n -d. Konstruksi pohon k-d mirip dengan konstruksi pohon biner seimbang satu dimensi. Kami mulai dengan sekumpulan contoh dan pada simpul akar kami membaginya sepanjang dimensi ke- i dengan menguji apakah $x_i \leq m$. Kami memilih nilai m sebagai median dari contoh-contoh sepanjang dimensi ke- i ; dengan demikian, setengah dari contoh akan berada di cabang kiri pohon dan setengah di kanan. Kami kemudian secara rekursif membuat pohon untuk kumpulan contoh kiri dan kanan, berhenti ketika ada kurang dari dua contoh yang tersisa. Untuk memilih dimensi untuk dibagi pada setiap simpul pohon, seseorang cukup memilih dimensi $i \bmod n$ pada tingkat i pohon. (Perhatikan bahwa kita mungkin perlu membagi pada dimensi apa pun beberapa kali saat kita melanjutkan ke bawah pohon.) Strategi lain adalah membagi pada dimensi yang memiliki sebaran nilai terluas.



Gambar 9.27 Kutukan dimensionalitas: (a) Panjang rata-rata lingkungan untuk 10 tetangga terdekat dalam hiperkubus satuan dengan 1.000.000 titik, sebagai fungsi dari jumlah dimensi. (b) Proporsi titik yang berada dalam kulit tipis yang terdiri dari 1% terluar

hiperkubus, sebagai fungsi dari jumlah dimensi. Diambil dari 10.000 titik yang terdistribusi secara acak.

Pencarian tepat dari pohon k-d sama seperti pencarian dari pohon biner (dengan sedikit komplikasi yang perlu Anda perhatikan yang mana dimensi yang Anda uji di setiap node). Namun, pencarian tetangga terdekat lebih rumit. Saat kita menelusuri cabang-cabang, membagi contoh menjadi dua, dalam beberapa kasus kita dapat membuang separuh contoh lainnya. Namun tidak selalu. Terkadang titik yang kita cari berada sangat dekat dengan batas pemisah. Titik kueri itu sendiri mungkin berada di sisi kiri batas, tetapi satu atau lebih dari k tetangga terdekat mungkin sebenarnya berada di sisi kanan. Kita harus menguji kemungkinan ini dengan menghitung jarak titik kueri ke batas pemisah, lalu mencari di kedua sisi jika kita tidak dapat menemukan k contoh di sebelah kiri yang lebih dekat dari jarak ini. Karena masalah ini, pohon k-d hanya sesuai jika terdapat lebih banyak contoh daripada dimensi, sebaiknya setidaknya 2^n contoh. Dengan demikian, pohon k-d bekerja dengan baik hingga 10 dimensi dengan ribuan contoh atau hingga 20 dimensi dengan jutaan contoh. Jika kita tidak memiliki cukup contoh, pencarian tidak lebih cepat daripada pemindaian linier dari seluruh set data.

Hashing yang Sensitif terhadap Lokalitas

Tabel hash berpotensi menyediakan pencarian yang bahkan lebih cepat daripada pohon biner. Namun, bagaimana kita dapat menemukan tetangga terdekat menggunakan tabel hash, jika kode hash bergantung pada *kecocokan* persis? Kode hash mendistribusikan nilai secara acak di antara bin, tetapi kita ingin titik-titik terdekat dikelompokkan bersama dalam bin yang sama; kita menginginkan **hash yang sensitif terhadap lokalitas** (LSH).

Kita tidak dapat menggunakan hash untuk menyelesaikan $NN(k, \mathbf{x}_q)$ secara eksak, tetapi dengan penggunaan algoritma acak yang cerdas, kita dapat menemukan solusi aproksimasi. Pertama, kita mendefinisikan masalah **tetangga dekat aproksimasi**: diberikan kumpulan data titik contoh dan titik kueri $\mathbf{x}_q$, temukan, dengan probabilitas tinggi, satu titik contoh (atau beberapa titik) yang dekat dengan $\mathbf{x}_q$. Untuk lebih tepat, kita mensyaratkan bahwa jika terdapat titik $\mathbf{x}_j$ yang berada dalam radius r dari $\mathbf{x}_q$, maka dengan probabilitas tinggi algoritma akan menemukan titik $\mathbf{x}_j$ yang berada dalam jarak cr dari q . Jika tidak ada titik dalam radius r , maka algoritma diperbolehkan untuk melaporkan kegagalan. Nilai c dan "probabilitas tinggi" adalah parameter algoritma.

Untuk menyelesaikan tetangga dekat aproksimasi, kita memerlukan fungsi hash $g(\mathbf{x})$ yang memiliki sifat bahwa, untuk setiap dua titik $\mathbf{x}_j$ dan $\mathbf{x}_{j'}$, probabilitas keduanya memiliki kode hash yang sama adalah kecil jika jaraknya lebih besar dari cr , dan tinggi jika jaraknya kurang dari r . Untuk menyederhanakannya, kita akan memperlakukan setiap titik sebagai string bit. (Fitur apa pun yang bukan Boolean dapat dikodekan menjadi sekumpulan fitur Boolean.)

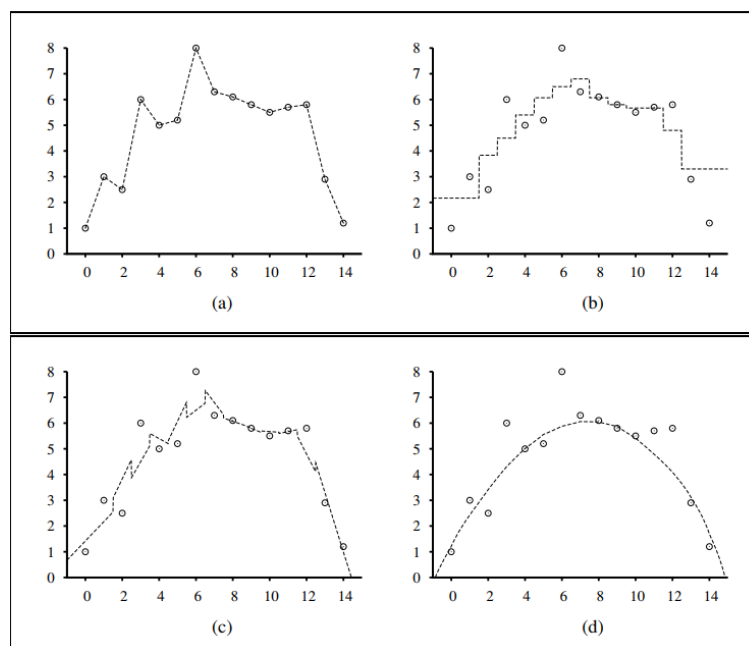
Intuisi yang kami andalkan adalah jika dua titik berdekatan dalam ruang n -dimensi, maka keduanya pasti akan berdekatan ketika diproyeksikan ke ruang satu dimensi (garis). Faktanya, kita dapat mendiskritkan garis tersebut ke dalam bin—keranjang hash—sehingga, dengan probabilitas tinggi, titik-titik yang berdekatan diproyeksikan ke bin yang sama persis.

Titik-titik yang berjauhan akan cenderung diproyeksikan ke bin yang berbeda untuk sebagian besar proyeksi, tetapi akan selalu ada beberapa proyeksi yang secara kebetulan memproyeksikan titik-titik yang berjauhan ke dalam bin yang sama. Dengan demikian, bin untuk titik $\mathbf{x}_q$ berisi banyak (tetapi tidak semua) titik yang dekat dengan $\mathbf{x}_q$, serta beberapa titik yang berjauhan.

Trik LSH adalah membuat *beberapa* proyeksi acak dan menggabungkannya. Proyeksi acak hanyalah subset acak dari representasi bit-string. Kami memilih ℓ proyeksi acak yang berbeda dan membuat ℓ tabel hash, $g_1(\mathbf{x}), \dots, g_\ell(\mathbf{x})$. Kami kemudian memasukkan semua contoh ke dalam setiap tabel hash. Kemudian, ketika diberikan titik kueri $\mathbf{x}_q$, kami mengambil himpunan titik dalam bin $g_k(q)$ untuk setiap k , dan menggabungkan himpunan ini menjadi himpunan titik kandidat, C . Kemudian, kami menghitung jarak aktual ke $\mathbf{x}_q$ untuk setiap titik di C dan mengembalikan k titik terdekat. Dengan probabilitas tinggi, setiap titik yang dekat dengan $\mathbf{x}_q$ akan muncul di setidaknya satu bin, dan meskipun beberapa titik yang jauh juga akan muncul, kami dapat mengabaikannya. Dengan permasalahan dunia nyata yang besar, seperti menemukan tetangga dekat dalam kumpulan data 13 juta gambar web menggunakan 512 dimensi, hashing yang peka terhadap lokalitas hanya perlu memeriksa beberapa ribu gambar dari 13 juta gambar untuk menemukan tetangga terdekat; peningkatan kecepatan seribu kali lipat dibandingkan pendekatan pohon ekshaustif atau k-d.

Regresi Nonparametrik

Sekarang kita akan membahas pendekatan nonparametrik untuk *regresi*, alih-alih klasifikasi. Gambar 9.28 menunjukkan contoh beberapa model yang berbeda. Pada (a), kita mungkin memiliki metode paling sederhana, yang secara informal dikenal sebagai "hubungkan titik-titik", dan secara angkuh disebut "regresi nonparametrik linier sepotong-sepotong". Model ini menciptakan fungsi $h(x)$ yang, ketika diberikan kueri x_q , menyelesaikan masalah regresi linier biasa hanya dengan dua titik: contoh pelatihan tepat di sebelah kiri dan kanan x_q . Ketika derau rendah, metode sepele ini sebenarnya tidak terlalu buruk, sehingga menjadi fitur standar perangkat lunak grafik dalam spreadsheet.

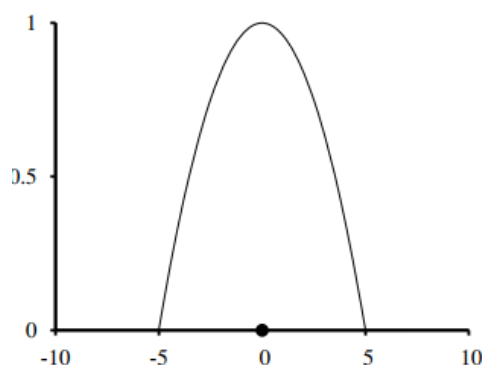


Gambar 9.28 Model regresi nonparametrik: (a) menghubungkan titik-titik, (b) rata-rata 3 tetangga terdekat, (c) regresi linier 3 tetangga terdekat, (d) regresi tertimbang lokal dengan kernel kuadratik dengan lebar $k = 10$.

Namun, ketika datanya berisik, fungsi yang dihasilkan bersifat runcing, dan tidak tergeneralisasi dengan baik.

Regresi k – tetangga terdekat (Gambar 9.28(b)) menyempurnakan metode hubungan-titik. Alih-alih hanya menggunakan dua contoh di sebelah kiri dan kanan titik kueri x_q , kami menggunakan k tetangga terdekat (di sini 3). Nilai k yang lebih besar cenderung menghaluskan besarnya lonjakan, meskipun fungsi yang dihasilkan memiliki diskontinuitas. Pada (b), kita memiliki rata-rata k -tetangga-terdekat: $h(x)$ adalah nilai rata-rata dari k titik, $\sum y_j / k$. Perhatikan bahwa pada titik-titik terluar, dekat $x = 0$ dan $x = 14$, estimasinya buruk karena semua bukti berasal dari satu sisi (sisi dalam), dan mengabaikan tren. Pada (c), kita memiliki regresi linier k -tetangga-terdekat, yang menemukan garis terbaik melalui k contoh. Ini lebih baik dalam menangkap tren pada outlier, tetapi masih terputus-putus. Pada (b) dan (c), kita dihadapkan pada pertanyaan tentang bagaimana memilih nilai k yang baik. Jawabannya, seperti biasa, adalah validasi silang.

Regresi terbobot lokal (Gambar 9.28(d)) memberi kita keuntungan dari tetangga terdekat, tanpa diskontinuitas. Untuk menghindari diskontinuitas dalam $h(x)$, kita perlu menghindari diskontinuitas dalam himpunan contoh yang kita gunakan untuk menaksir $h(x)$. Ide regresi terbobot lokal adalah bahwa pada setiap titik kueri x_q , contoh yang dekat dengan x_q diberi bobot yang besar, dan contoh yang lebih jauh diberi bobot yang lebih kecil atau bahkan tidak diberi bobot sama sekali. Penurunan bobot terhadap jarak selalu bertahap, tidak tiba-tiba.



Gambar 9.29 Kernel kuadrat, $K(d) = \max(0, 1 - (2|x|/k)^2)$, dengan lebar kernel $k = 10$, berpusat pada titik kueri $x = 0$.

Kita memutuskan seberapa besar bobot untuk setiap contoh dengan fungsi yang dikenal sebagai **kernel**. Fungsi kernel tampak seperti tonjolan; pada Gambar 9.29 kita melihat kernel spesifik yang digunakan untuk menghasilkan Gambar 9.28(d). Kita dapat melihat bahwa bobot yang diberikan oleh kernel ini paling tinggi di bagian tengah dan mencapai nol pada jarak ± 5 . Bisakah kita memilih sembarang fungsi untuk sebuah kernel? Tidak. Pertama, perhatikan bahwa kita memanggil fungsi kernel K dengan $K(\text{Distance}(\mathbf{x}_j, \mathbf{x}_q))$, di mana $\mathbf{x}_q$ adalah titik kueri yang berjarak tertentu dari $\mathbf{x}_j$, dan kita ingin tahu berapa banyak bobot yang harus diberikan pada jarak tersebut. Jadi K harus simetris di sekitar 0 dan memiliki nilai maksimum di 0. Area di bawah kernel harus tetap dibatasi saat kita menuju ke $\pm\infty$. Bentuk lain, seperti Gaussian, telah digunakan untuk kernel, tetapi penelitian terbaru menunjukkan bahwa pilihan bentuk tidak terlalu penting. Kita harus berhati-hati tentang lebar kernel. Sekali lagi, ini adalah parameter model yang paling baik dipilih dengan validasi silang. Sama seperti dalam memilih k untuk tetangga terdekat, jika kernel terlalu lebar kita akan mendapatkan underfitting dan jika terlalu sempit kita akan mendapatkan overfitting. Pada Gambar 9.29(d), nilai $k = 10$ menghasilkan kurva halus yang terlihat tepat—tetapi mungkin kurva tersebut tidak cukup memperhatikan outlier pada $x = 6$; lebar kernel yang lebih sempit akan lebih responsif terhadap titik-titik individual.

Melakukan regresi terbobot lokal dengan kernel kini menjadi mudah. Untuk titik kueri $\mathbf{x}_q$ yang diberikan, kami memecahkan masalah regresi terbobot berikut menggunakan penurunan gradien:

$$w^* = \underset{w}{\operatorname{argmin}} \sum_j K(\text{Distance}(x_q, x_j)) (y_j - w \cdot x_j)^2,$$

dengan *Jarak* adalah salah satu metrik jarak yang dibahas untuk tetangga terdekat. Maka jawabannya adalah $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$.

Perhatikan bahwa kita perlu menyelesaikan masalah regresi baru untuk *setiap* titik kueri—itulah arti *lokal*. (Dalam regresi linier biasa, kita menyelesaikan masalah regresi sekali,

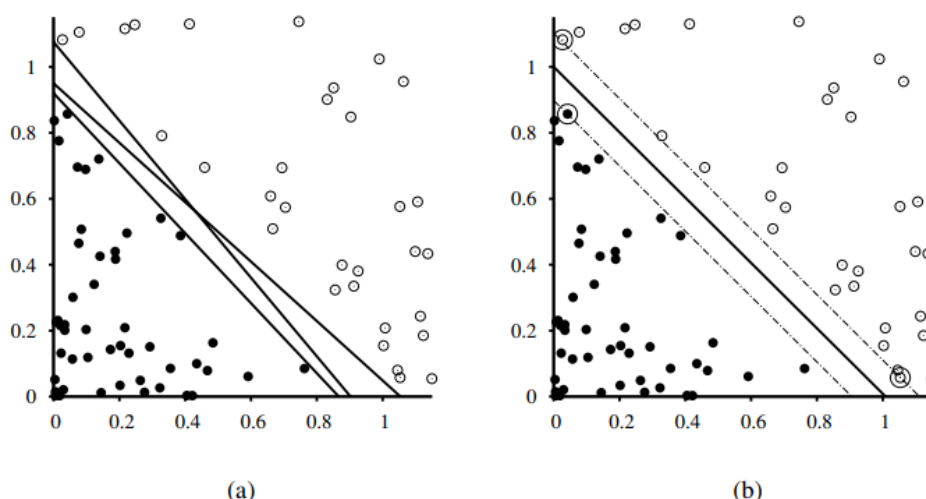
secara global, lalu menggunakan h_w yang sama untuk setiap titik kueri.) Yang mengurangi pekerjaan tambahan ini adalah kenyataan bahwa setiap masalah regresi akan lebih mudah dipecahkan, karena hanya melibatkan contoh dengan bobot bukan nol—contoh yang kernelnya tumpang tindih dengan titik kueri. Ketika lebar kernel kecil, ini mungkin hanya beberapa titik.

Sebagian besar model nonparametrik memiliki keuntungan karena mudah melakukan validasi silang tanpa perlu menghitung ulang semuanya. Dengan model k -tetangga-terdekat, misalnya, ketika diberikan contoh uji $(\mathbf{x}, y)$, kami mengambil k tetangga terdekat satu kali, menghitung kerugian per contoh $L(y, h(\mathbf{x}))$ dari mereka, dan mencatatnya sebagai hasil penghapusan satu untuk setiap contoh yang bukan salah satu tetangga. Kemudian, kami mengambil $k + 1$ tetangga terdekat dan mencatat hasil yang berbeda untuk setiap k tetangga yang dihapus. Dengan N contoh, seluruh proses adalah $O(k)$, bukan $O(kN)$.

9.9 MESIN VEKTOR PENDUKUNG

Mesin vektor pendukung atau kerangka kerja SVM saat ini merupakan pendekatan paling populer untuk pembelajaran terawasi "siapa pakai": jika Anda tidak memiliki pengetahuan khusus sebelumnya tentang suatu domain, maka SVM adalah metode yang sangat baik untuk dicoba terlebih dahulu. Ada tiga sifat yang membuat SVM menarik:

1. SVM membangun **pemisah margin maksimum**—batas keputusan dengan jarak terbesar yang mungkin ke titik-titik contoh. Hal ini membantu mereka menggeneralisasi dengan baik.
2. SVM menciptakan hiperbidang pemisah linier, tetapi mereka memiliki kemampuan untuk menanamkan data ke dalam ruang berdimensi lebih tinggi, menggunakan apa yang disebut **trik kernel**. Seringkali, data yang tidak dapat dipisahkan secara linier di ruang masukan asli dapat dengan mudah dipisahkan di ruang berdimensi lebih tinggi. Pemisah linier berdimensi tinggi sebenarnya nonlinier di ruang asli. Ini berarti ruang hipotesis jauh lebih luas daripada metode yang menggunakan representasi linier murni.
3. SVM adalah metode nonparametrik—mereka menyimpan contoh pelatihan dan berpotensi perlu menyimpan semuanya. Di sisi lain, dalam praktiknya, SVM seringkali hanya menyimpan sebagian kecil dari jumlah contoh—terkadang hanya sejumlah kecil konstanta dikalikan jumlah dimensi. Dengan demikian, SVM menggabungkan keunggulan model nonparametrik dan parametrik: model ini memiliki fleksibilitas untuk merepresentasikan fungsi kompleks, tetapi tahan terhadap overfitting.



Gambar 9.30 Klasifikasi mesin vektor pendukung: (a) Dua kelas titik (lingkaran hitam dan putih) dan tiga kandidat pemisah linier. (b) Pemisah margin maksimum (garis tebal) berada di titik tengah margin (area di antara garis putus-putus). Vektor pendukung (titik dengan lingkaran besar) adalah contoh yang paling dekat dengan pemisah.

Bisa dikatakan bahwa SVM berhasil karena satu wawasan utama dan satu trik yang rapi. Kita akan membahas masing-masing secara bergantian. Pada Gambar 9.30(a), kita memiliki masalah klasifikasi biner dengan tiga batas keputusan kandidat, masing-masing merupakan pemisah linier. Masing-masing konsisten dengan semua contoh, sehingga dari sudut pandang kerugian 0/1, masing-masing akan sama baiknya. Regresi logistik akan menemukan beberapa garis pemisah; lokasi pasti garis tersebut bergantung pada *semua* titik contoh. Wawasan utama SVM adalah bahwa beberapa contoh lebih penting daripada yang lain, dan bahwa memperhatikannya dapat menghasilkan generalisasi yang lebih baik.

Perhatikan garis pemisah terendah dari tiga garis pemisah pada (a). Garis tersebut sangat mendekati 5 contoh hitam. Meskipun mengklasifikasikan semua contoh dengan benar, dan dengan demikian meminimalkan kerugian, Anda seharusnya merasa khawatir karena begitu banyak contoh yang mendekati garis; mungkin saja contoh hitam lainnya akan berada di sisi lain garis.

SVM mengatasi masalah ini: Alih-alih meminimalkan *kerugian empiris* yang diharapkan pada data pelatihan, SVM mencoba meminimalkan *kerugian generalisasi* yang diharapkan. Kita tidak tahu di mana titik-titik yang belum terlihat itu mungkin berada, tetapi dengan asumsi probabilistik bahwa titik-titik tersebut diambil dari distribusi yang sama dengan contoh-contoh yang telah terlihat sebelumnya, ada beberapa argumen dari teori pembelajaran komputasional yang menyarankan agar kita meminimalkan kerugian generalisasi dengan memilih pemisah yang terjauh dari contoh-contoh yang telah kita lihat sejauh ini. Kita menyebut pemisah ini, yang ditunjukkan pada Gambar 9.30(b), sebagai **pemisah margin maksimum**. **Margin** adalah lebar area yang dibatasi oleh garis putus-putus pada gambar—dua kali jarak dari pemisah ke titik contoh terdekat.

Sekarang, bagaimana kita menemukan pemisah ini? Sebelum menunjukkan persamaan, beberapa notasi: Secara tradisional SVM menggunakan konvensi bahwa label kelas adalah +1 dan -1, alih-alih +1 dan 0 yang telah kita gunakan sejauh ini. Selain itu, ketika kita menempatkan intersep ke dalam vektor bobot $\mathbf{w}$ (dan nilai dummy 1 yang sesuai ke dalam $x_{j,0}$), SVM tidak melakukan hal itu; mereka menyimpan intersep sebagai parameter terpisah, b . Dengan mengingat hal itu, pemisah didefinisikan sebagai himpunan titik $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$. Kita dapat mencari ruang $\mathbf{w}$ dan b dengan penurunan gradien untuk menemukan parameter yang memaksimalkan margin sambil mengklasifikasikan semua contoh dengan benar.

Namun, ternyata ada pendekatan lain untuk menyelesaikan masalah ini. Kami tidak akan menunjukkan detailnya, tetapi hanya akan mengatakan bahwa ada representasi alternatif yang disebut representasi dual, di mana solusi optimal ditemukan dengan menyelesaikan

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (x_j \cdot x_k) \quad (9.13)$$

dengan batasan $\alpha_j \geq 0$ dan $\sum_j j \alpha_j y_j = 0$. Ini adalah masalah optimasi **pemrograman kuadrat**, yang dapat diselesaikan dengan paket perangkat lunak yang baik. Setelah kita menemukan vektor α , kita dapat kembali ke $\mathbf{w}$ dengan persamaan $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$, atau kita dapat tetap menggunakan representasi dual. Ada tiga sifat penting dari Persamaan (9.13). Pertama, ekspresi tersebut cembung; memiliki satu nilai maksimum global yang dapat ditemukan secara efisien. Kedua, *data yang masuk ke dalam ekspresi hanya dalam bentuk perkalian titik dari pasangan titik*. Sifat kedua ini juga berlaku untuk persamaan pemisah itu sendiri; setelah nilai α_j optimal dihitung,

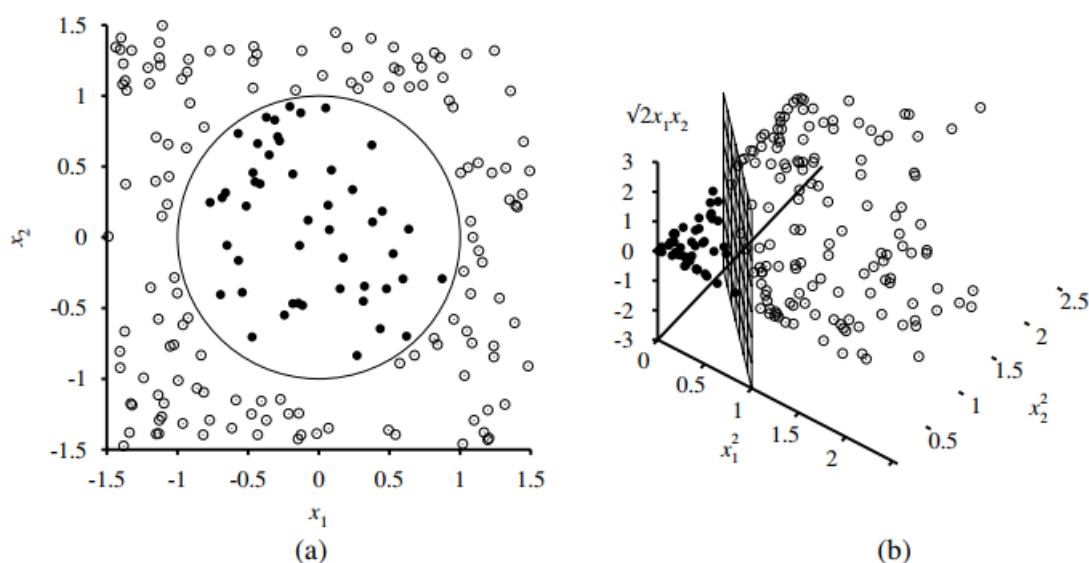
$$h(x) = \operatorname{sign} \left(\sum_j \alpha_j y_j (x \cdot x_j) - b \right). \quad (9.14)$$

Properti penting terakhir adalah bobot α_j yang terkait dengan setiap titik data adalah *not*, kecuali untuk **vektor pendukung**—titik-titik terdekat dengan pemisah. (Vektor-vektor ini disebut vektor "pendukung" karena "menahan" bidang pemisah.) Karena biasanya terdapat lebih sedikit vektor pendukung daripada contoh, SVM mendapatkan beberapa keuntungan dari model parametrik.

Bagaimana jika contoh-contoh tersebut tidak dapat dipisahkan secara linear? Gambar 9.31(a) menunjukkan ruang masukan yang didefinisikan oleh atribut $\mathbf{x} = (x_1, x_2)$, dengan contoh positif ($y = +1$) di dalam wilayah lingkaran dan contoh negatif ($y = -1$) di luar. Jelas, tidak ada pemisah linear untuk masalah ini. Sekarang, misalkan kita mengekspresikan ulang data masukan—yaitu, kita memetakan setiap vektor masukan $\mathbf{x}$ ke vektor baru dengan nilai fitur, $F(\mathbf{x})$. Secara khusus, mari kita gunakan tiga fitur tersebut.

$$f_1 = x_1^2, \quad f_2 = x_2^2, \quad f_3 = \sqrt{2}x_1x_2. \quad (9.15)$$

Kita akan segera melihat dari mana asalnya, tetapi untuk saat ini, lihat saja apa yang terjadi. Gambar 9.31(b) menunjukkan data dalam ruang tiga dimensi baru yang didefinisikan oleh tiga fitur; data dapat *dipisahkan secara linear* dalam ruang ini! Fenomena ini sebenarnya cukup umum: jika data dipetakan ke dalam ruang berdimensi cukup tinggi, maka data tersebut hampir selalu dapat dipisahkan secara linear—jika Anda melihat sekumpulan titik dari berbagai arah, Anda akan menemukan cara untuk menyelaraskannya. Di sini, kami hanya menggunakan tiga dimensi;



Gambar 9.31 (a) Set pelatihan dua dimensi dengan contoh positif berupa lingkaran hitam dan contoh negatif berupa lingkaran putih. Batas keputusan sebenarnya, $x_1^2 + x_2^2 \leq 1$, juga ditunjukkan. (b) Data yang sama setelah dipetakan ke dalam ruang masukan tiga dimensi $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. Batas keputusan melingkar pada (a) menjadi batas keputusan linear dalam tiga dimensi. Gambar 9.30(b) memberikan tampilan dekat pemisah pada (b).

Biasanya, kita tidak akan menemukan pemisah linear di ruang masukan $\mathbf{x}$, tetapi kita dapat menemukan pemisah linear di ruang fitur berdimensi tinggi $F(\mathbf{x})$ hanya dengan mengganti $\mathbf{x}_j \cdot \mathbf{x}_k$ pada Persamaan (9.13) dengan $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$. Hal ini sendiri tidaklah luar biasa—mengganti $\mathbf{x}$ dengan $F(\mathbf{x})$ dalam algoritma pembelajaran *apapun* memang memiliki efek yang dibutuhkan—tetapi perkalian titik memiliki beberapa sifat khusus. Ternyata $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ seringkali dapat dihitung tanpa terlebih dahulu menghitung F untuk setiap titik. Dalam ruang fitur tiga dimensi kita yang didefinisikan oleh Persamaan (9.15), sedikit aljabar menunjukkan bahwa

$$F(\mathbf{x}_j) \cdot F(\mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2.$$

(Itulah mengapa angka $\sqrt{2}$ ada di f_3 .) Ekspresi $(\mathbf{x}_j \cdot \mathbf{x}_k)^2$ disebut **fungsi kernel**,¹² dan biasanya ditulis sebagai $k(\mathbf{x}_j \cdot \mathbf{x}_k)$. Fungsi kernel dapat diterapkan pada pasangan data masukan untuk mengevaluasi produk titik dalam beberapa ruang fitur yang sesuai. Jadi, kita dapat menemukan pemisah linear dalam ruang fitur berdimensi lebih tinggi $F(\mathbf{x})$ hanya dengan mengganti $\mathbf{x}_j \cdot \mathbf{x}_k$ dalam Persamaan (9.13) dengan fungsi kernel $k(\mathbf{x}_j \cdot \mathbf{x}_k)$. Dengan demikian, kita dapat belajar dalam ruang berdimensi lebih tinggi, tetapi kita hanya menghitung fungsi kernel, bukan daftar lengkap fitur untuk setiap titik data.

Langkah selanjutnya adalah melihat bahwa tidak ada yang istimewa tentang $k(\mathbf{x}_j \cdot \mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2$. Ini sesuai dengan ruang fitur berdimensi lebih tinggi tertentu, tetapi fungsi kernel lainnya sesuai dengan ruang fitur lainnya. Sebuah hasil yang sangat dihormati dalam matematika, **teorema Mercer** (1909), memberi tahu kita bahwa setiap fungsi kernel yang "masuk akal"¹³ berkorespondensi dengan *suatu* ruang fitur. Ruang fitur ini bisa sangat besar, bahkan untuk kernel yang tampak tidak berbahaya. Misalnya, **kernel polinomial**, $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$, berkorespondensi dengan ruang fitur yang dimensinya eksponensial dalam d .

Inilah **trik kernel** yang cerdas: Dengan memasukkan kernel-kernel ini ke Persamaan (9.13), *pemisah linier optimal dapat ditemukan secara efisien dalam ruang fitur dengan miliaran (atau, dalam beberapa kasus, tak terhingga banyaknya) dimensi*. Pemisah linier yang dihasilkan, ketika dipetakan kembali ke ruang masukan asli, dapat berkorespondensi dengan batas keputusan nonlinier yang berfluktuasi secara sembarangan antara contoh positif dan negatif.

Dalam kasus data yang secara inheren berisik, kita mungkin tidak menginginkan pemisah linier di ruang berdimensi tinggi. Sebaliknya, kita menginginkan permukaan keputusan di ruang berdimensi lebih rendah yang tidak memisahkan kelas-kelas secara jelas, tetapi mencerminkan realitas data yang berisik. Hal itu dimungkinkan dengan pengklasifikasi **margin lunak**, yang memungkinkan contoh-contoh berada di sisi yang salah dari batas keputusan, tetapi memberikan penalti yang proporsional dengan jarak yang diperlukan untuk memindahkannya kembali ke sisi yang benar.

Metode kernel dapat diterapkan tidak hanya dengan algoritma pembelajaran yang menemukan pemisah linear optimal, tetapi juga dengan algoritma lain yang dapat diformulasikan ulang agar hanya bekerja dengan perkalian titik dari pasangan titik data, seperti pada Persamaan 9.13 dan 9.14. Setelah ini selesai, perkalian titik digantikan oleh fungsi kernel dan kita memiliki versi algoritma yang telah **dikernelisasi**. Hal ini dapat dilakukan dengan mudah untuk pembelajaran k-nearest-neighbors dan perceptron, antara lain.

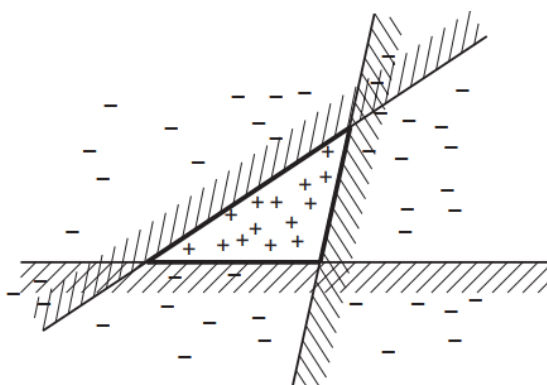
9.10 PEMBELAJARAN ENSEMBLE

Sejauh ini kita telah membahas metode pembelajaran yang menggunakan satu hipotesis tunggal, yang dipilih dari ruang hipotesis, untuk membuat prediksi. Ide di balik metode **pembelajaran ansambel** adalah memilih sekumpulan, atau ansambel, hipotesis dari ruang hipotesis dan **menggabungkan** prediksinya. Misalnya, selama validasi silang, kita dapat

menghasilkan dua puluh pohon keputusan yang berbeda, dan meminta mereka untuk memilih klasifikasi terbaik untuk contoh baru.

Motivasi pembelajaran ansambel sederhana. Pertimbangkan ansambel dengan $K = 5$ hipotesis dan anggaplah kita menggabungkan prediksi mereka menggunakan pemungutan suara mayoritas sederhana. Agar ansambel salah mengklasifikasikan contoh baru, *setidaknya tiga dari lima hipotesis harus salah mengklasifikasikannya*. Harapannya adalah kemungkinan ini jauh lebih kecil daripada kesalahan klasifikasi oleh satu hipotesis. Anggaplah kita mengasumsikan bahwa setiap hipotesis h_k dalam ansambel memiliki kesalahan p —yaitu, probabilitas bahwa contoh yang dipilih secara acak salah diklasifikasikan oleh h_k adalah p . Lebih lanjut, anggaplah kita mengasumsikan bahwa kesalahan yang dibuat oleh setiap hipotesis bersifat independen. Dalam kasus tersebut, jika p kecil, maka probabilitas terjadinya sejumlah besar kesalahan klasifikasi sangat kecil. Misalnya, perhitungan sederhana menunjukkan bahwa penggunaan ansambel lima hipotesis mengurangi tingkat kesalahan 1 banding 10 menjadi tingkat kesalahan kurang dari 1 banding 100. Tentu saja, asumsi independensi ini tidak masuk akal, karena hipotesis cenderung disesatkan dengan cara yang sama oleh aspek data pelatihan yang menyesatkan. Namun, jika hipotesis-hipotesis tersebut setidaknya sedikit berbeda, sehingga mengurangi korelasi antara kesalahan-kesalahannya, maka pembelajaran ansambel dapat sangat berguna.

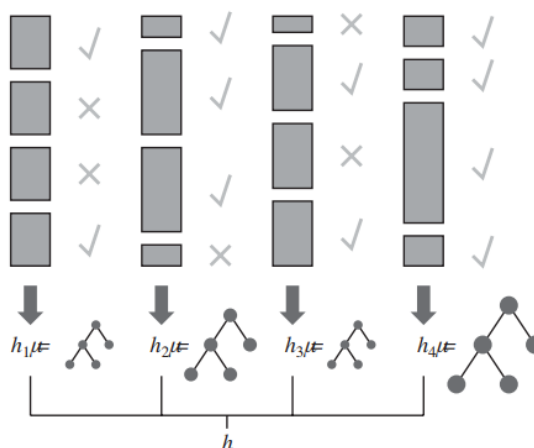
Cara lain untuk memikirkan gagasan ansambel adalah sebagai cara umum untuk memperluas ruang hipotesis. Artinya, anggaplah ansambel itu sendiri sebagai sebuah hipotesis dan ruang hipotesis baru sebagai himpunan semua ansambel yang mungkin yang dapat dikonstruksi dari hipotesis-hipotesis di ruang asal. Gambar 9.32 menunjukkan bagaimana hal ini dapat menghasilkan ruang hipotesis yang lebih ekspresif. Jika ruang hipotesis asli memungkinkan algoritma pembelajaran yang sederhana dan efisien, maka metode ansambel menyediakan cara untuk mempelajari kelas hipotesis yang jauh lebih ekspresif tanpa menimbulkan banyak kerumitan komputasi atau algoritmik tambahan.



Gambar 9.32 Ilustrasi peningkatan daya ekspresif yang diperoleh melalui pembelajaran ansambel. Kami mengambil tiga hipotesis ambang batas linear, yang masing-masing terklasifikasi positif pada sisi yang tidak diarsir, dan mengklasifikasikan sebagai positif setiap contoh yang terklasifikasi positif oleh ketiganya. Wilayah segitiga yang dihasilkan merupakan hipotesis yang tidak dapat diekspresikan dalam ruang hipotesis asli.

Metode ensemble yang paling banyak digunakan disebut **boosting**. Untuk memahami cara kerjanya, pertama-tama kita perlu menjelaskan konsep **set pelatihan berbobot**. Dalam set pelatihan seperti itu, setiap contoh memiliki bobot terkait $w_j \geq 0$. Semakin tinggi bobot suatu contoh, semakin tinggi pula kepentingan yang melekat padanya selama pembelajaran hipotesis. Sangat mudah untuk memodifikasi algoritma pembelajaran yang telah kita lihat sejauh ini agar dapat beroperasi dengan set pelatihan berbobot.

Boosting dimulai dengan $w_j = 1$ untuk semua contoh (yaitu, set pelatihan normal). Dari set ini, dihasilkan hipotesis pertama, h_1 . Hipotesis ini akan mengklasifikasikan beberapa contoh pelatihan dengan benar dan beberapa salah. Kita ingin hipotesis berikutnya bekerja lebih baik pada contoh yang salah klasifikasi, jadi kita meningkatkan bobotnya sambil mengurangi bobot contoh yang diklasifikasikan dengan benar. Dari set pelatihan berbobot baru ini, kita menghasilkan hipotesis h_2 . Proses ini berlanjut dengan cara ini hingga kita telah menghasilkan K hipotesis, di mana K adalah input untuk algoritma boosting. Hipotesis ensemble akhir adalah kombinasi mayoritas terbobot dari semua K hipotesis, masing-masing dibobot berdasarkan seberapa baik kinerjanya pada set pelatihan. Gambar 9.33 menunjukkan cara kerja algoritma secara konseptual. Ada banyak varian dari ide dasar boosting, dengan berbagai cara untuk menyesuaikan bobot dan menggabungkan hipotesis. Satu algoritma spesifik, yang disebut ADABOOST, ditunjukkan pada Gambar 9.34. ADABOOST memiliki properti yang sangat penting: jika algoritma pembelajaran masukan L adalah algoritma **pembelajaran yang lemah**—yang berarti L selalu mengembalikan hipotesis dengan akurasi pada set pelatihan yang sedikit lebih baik daripada tebakan acak (yaitu, $50\% + \epsilon$ untuk klasifikasi Boolean)—maka ADABOOST akan mengembalikan hipotesis yang *mengklasifikasikan data pelatihan dengan sempurna* untuk K yang cukup besar. Dengan demikian, algoritma ini *meningkatkan* akurasi algoritma pembelajaran asli pada data pelatihan. Hasil ini berlaku tidak peduli seberapa tidak ekspresif ruang hipotesis asli dan tidak peduli seberapa kompleks fungsi yang sedang dipelajari.



Gambar 9.33 Cara kerja algoritma boosting. Setiap persegi panjang yang diarsir sesuai dengan sebuah contoh; tinggi persegi panjang sesuai dengan bobotnya. Tanda centang dan

tanda silang menunjukkan apakah contoh tersebut diklasifikasikan dengan benar oleh hipotesis saat ini. Ukuran pohon keputusan menunjukkan bobot hipotesis tersebut dalam ansambel akhir.

Mari kita lihat seberapa baik boosting pada data restoran. Kami akan memilih kelas **puntung keputusan**, yang merupakan pohon keputusan dengan hanya satu pengujian, sebagai ruang hipotesis awal kami. Kurva bawah pada Gambar 9.35(a) menunjukkan bahwa tunggal keputusan yang tidak diboost tidak terlalu efektif untuk set data ini, hanya mencapai kinerja prediksi 81% pada 100 contoh pelatihan. Ketika boosting diterapkan (dengan $K = 5$), kinerjanya lebih baik, mencapai 93% setelah 100 contoh.

Hal menarik terjadi seiring bertambahnya ukuran ansambel K . Gambar 9.35(b) menunjukkan performa set pelatihan (pada 100 contoh) sebagai fungsi K . Perhatikan bahwa galat mencapai nol ketika K bernilai 20; artinya, kombinasi mayoritas tertimbang dari 20 tunggal keputusan cukup untuk mencocokkan 100 contoh secara tepat. Seiring bertambahnya tunggal ke ansambel, galat tetap nol. Grafik tersebut juga menunjukkan bahwa performa set *pengujian terus meningkat lama setelah galat set pelatihan mencapai nol*. Pada $K = 20$, performa pengujian adalah 0,95 (atau galat 0,05), dan performa meningkat menjadi 0,98 paling lambat pada $K = 137$, sebelum secara bertahap turun menjadi 0,95.

```

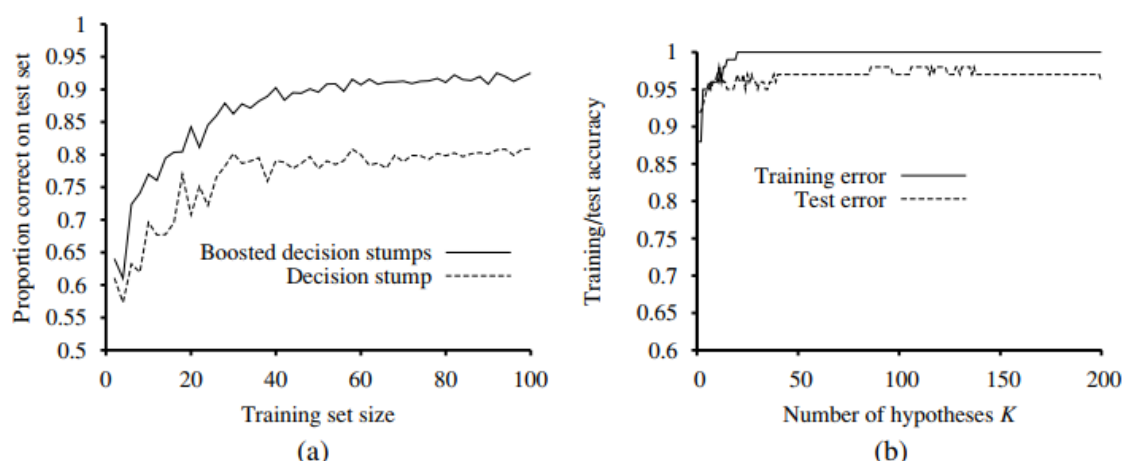
function ADABOOST(examples,  $L$ ,  $K$ ) returns a weighted – majority hypothesis
  inputs: examples, set of  $N$  labeled examples  $(x_1, y_1), \dots, (x_N, y_N)$ 
            $L$ , a learning algorithm
            $K$ , the number of hypotheses in the ensemble
  local variables: w, a vector of  $N$  example weights, initially  $1/N$ 
                  h, a vector of  $K$  hypotheses
                  z, a vector of  $K$  hypothesis weights

  for  $k = 1$  to  $K$  do
    h[ $k$ ]  $\leftarrow L(\text{examples}, \mathbf{w})$ 
    error  $\leftarrow 0$ 
    for  $j = 1$  to  $N$  do
      if h[ $k$ ]( $x_j$ )  $\neq y_j$  then error  $\leftarrow$  error +  $w[j]$ 
    for  $j = 1$  to  $N$  do
      if h[ $k$ ]( $x_j$ )  $= y_j$  then w[ $j$ ]  $\leftarrow$  w[ $j$ ]  $\cdot$  error / (1 – error)
    w  $\leftarrow$  NORMALIZE(w)
    z[ $k$ ]  $\leftarrow$  log (1 – error) / error
  return WEIGHTED – MAJORITY(h, z)

```

Gambar 9.34 Varian ADABOOST dari metode boosting untuk pembelajaran ensemble.

Algoritme ini menghasilkan hipotesis dengan melakukan pembobotan ulang secara berurutan pada contoh pelatihan. Fungsi WEIGHTED-MAJORITY menghasilkan hipotesis yang mengembalikan nilai keluaran dengan suara tertinggi dari hipotesis di h , dengan suara yang dibobotkan oleh z .



Gambar 9.35 (a) Grafik yang menunjukkan kinerja tunggal keputusan yang ditingkatkan dengan $K = 5$ versus tunggal keputusan yang tidak ditingkatkan pada data restoran. (b) Proporsi yang benar pada set pelatihan dan set pengujian sebagai fungsi K , jumlah hipotesis dalam ansambel. Perhatikan bahwa akurasi set pengujian sedikit meningkat bahkan setelah akurasi pelatihan mencapai 1, yaitu, setelah ansambel sesuai dengan data secara tepat.

Temuan ini, yang cukup robust di seluruh set data dan ruang hipotesis, cukup mengejutkan ketika pertama kali disadari. Pisau cukur Ockham memberi tahu kita untuk tidak membuat hipotesis lebih kompleks dari yang diperlukan, tetapi grafik menunjukkan bahwa prediksi *berubah* seiring dengan meningkatnya kompleksitas hipotesis ansambel! Berbagai penjelasan telah diajukan untuk hal ini. Salah satu pandangan adalah bahwa boosting mengaproksimasi **pembelajaran Bayesian**, yang dapat ditunjukkan sebagai algoritma pembelajaran yang optimal, dan aproksimasinya meningkat seiring dengan penambahan lebih banyak hipotesis. Penjelasan lain yang mungkin adalah bahwa penambahan hipotesis *lebih lanjut* memungkinkan ansambel untuk lebih tegas dalam membedakan antara contoh positif dan negatif, yang membantunya dalam mengklasifikasikan contoh baru.

Pembelajaran Daring

Sejauh ini, semua yang telah kita lakukan dalam bab ini bergantung pada asumsi bahwa data bersifat i.i.d. (independen dan terdistribusi identik). Di satu sisi, asumsi ini masuk akal: jika masa depan tidak memiliki kemiripan dengan masa lalu, lalu bagaimana kita bisa memprediksi apa pun? Di sisi lain, asumsi ini terlalu kuat: jarang masukan kita menangkap semua informasi yang akan membuat masa depan benar-benar independen dari masa lalu.

Di bagian ini, kita akan membahas apa yang harus dilakukan ketika data tidak bersifat i.i.d.; ketika data dapat berubah seiring waktu. Dalam hal ini, penting untuk membuat prediksi, jadi kita akan mengadopsi perspektif yang disebut pembelajaran daring: seorang agen menerima masukan x_j dari alam, memprediksi y_j yang sesuai, dan kemudian diberi tahu jawaban yang benar. Kemudian proses ini berulang dengan x_{j+1} , dan seterusnya. Kita mungkin

berpikir tugas ini sia-sia jika alam bersifat adversarial, semua prediksi mungkin salah. Ternyata ada beberapa jaminan yang bisa kita buat.

Mari kita pertimbangkan situasi di mana masukan kita terdiri dari prediksi dari panel pakar. Misalnya, setiap hari, sekumpulan K pakar memprediksi apakah pasar saham akan naik atau turun, dan tugas kita adalah mengumpulkan prediksi-prediksi tersebut dan membuat prediksi kita sendiri. Salah satu cara untuk melakukannya adalah dengan melacak seberapa baik kinerja setiap pakar, dan memilih untuk mempercayai mereka secara proporsional dengan kinerja mereka sebelumnya. Ini disebut algoritma mayoritas tertimbang acak. Kita dapat mendeskripsikannya secara lebih formal:

1. Inisialisasi sekumpulan bobot $\{w_1, \dots, w_K\}$ semuanya hingga 1.
2. Terima prediksi $\{\hat{y}_1, \dots, \hat{y}_K\}$ dari para pakar.
3. Pilih pakar k^* secara acak, secara proporsional dengan bobotnya: $P(k) = w_k / (\sum_{k'} w_{k'})$.
4. Prediksi $\hat{y}_{k^*}$.
5. Terima jawaban yang benar y .
6. Untuk setiap pakar k sehingga $\hat{y}_k \neq y$, perbarui $w_k \leftarrow \beta w_k$

Di sini, β adalah angka, $0 < \beta < 1$, yang menunjukkan seberapa besar penalti yang harus diberikan kepada seorang pakar untuk setiap kesalahan.

Kami mengukur keberhasilan algoritma ini dalam hal penyesalan, yang didefinisikan sebagai jumlah kesalahan tambahan yang kami buat dibandingkan dengan pakar yang, jika dipikir-pikir, memiliki catatan prediksi terbaik. Misalkan M^* adalah jumlah kesalahan yang dibuat oleh pakar terbaik. Maka, jumlah kesalahan, M , yang dibuat oleh algoritma mayoritas tertimbang acak, dibatasi oleh¹⁵

$$M < \frac{M^* \ln(1/\beta) + \ln K}{1 - \beta}.$$

Batasan ini berlaku untuk semua rangkaian contoh, bahkan yang dipilih oleh musuh yang mencoba melakukan yang terburuk. Untuk lebih spesifik, ketika ada $K = 10$ pakar, jika kita memilih $\beta = 1/2$ maka jumlah kesalahan kita dibatasi oleh $1,39M^* + 4,6$, dan jika $\beta = 3/4$ oleh $1,15M^* + 9,2$. Secara umum, jika β mendekati 1 maka kita responsif terhadap perubahan dalam jangka panjang; jika pakar terbaik berubah, kita akan menyadarinya sebelum terlalu lama.

Namun, kita membayar penalti di awal, ketika kita memulai dengan semua pakar yang dipercaya secara setara; kita mungkin menerima saran dari pakar yang buruk terlalu lama. Ketika β mendekati 0, kedua faktor ini terbalik. Perhatikan bahwa kita dapat memilih β untuk mendekati M^* secara asimtotik dalam jangka panjang; ini disebut pembelajaran tanpa penyesalan (karena jumlah rata-rata penyesalan per percobaan cenderung menjadi 0 seiring bertambahnya jumlah percobaan).

Pembelajaran daring bermanfaat ketika data mungkin berubah dengan cepat seiring waktu. Pembelajaran daring juga berguna untuk aplikasi yang melibatkan kumpulan data besar

yang terus bertambah, meskipun perubahannya bertahap. Misalnya, dengan basis data jutaan gambar web, Anda tentu tidak ingin melatih, katakanlah, model regresi linier pada semua data, lalu melatih ulang dari awal setiap kali gambar baru ditambahkan. Akan lebih praktis untuk memiliki algoritma daring yang memungkinkan gambar ditambahkan secara bertahap. Untuk sebagian besar algoritma pembelajaran yang didasarkan pada minimalisasi kerugian, terdapat versi daring yang didasarkan pada minimalisasi penyesalan. Merupakan nilai tambah bahwa banyak dari algoritma daring ini dilengkapi dengan batasan penyesalan yang terjamin.

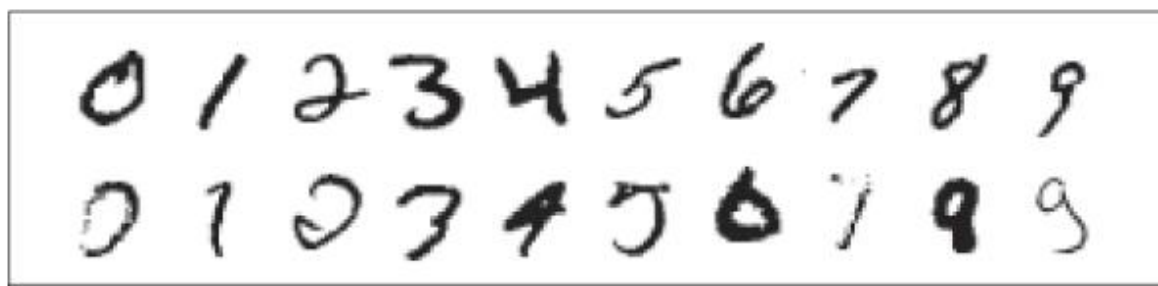
Bagi beberapa pengamat, sungguh mengejutkan bahwa ada batasan yang begitu ketat tentang seberapa baik kinerja kita dibandingkan dengan panel ahli. Bagi yang lain, hal yang benar-benar mengejutkan adalah ketika panel ahli manusia berkumpul memprediksi harga pasar saham, hasil pertandingan olahraga, atau kontes politik publik yang menonton begitu bersedia mendengarkan mereka berceramah dan begitu enggan untuk mengukur tingkat kesalahan mereka.

9.11 PEMBELAJARAN MESIN PRAKTIS

Kami telah memperkenalkan beragam teknik pembelajaran mesin, masing-masing diilustrasikan dengan tugas-tugas pembelajaran sederhana. Di bagian ini, kami mempertimbangkan dua aspek pembelajaran mesin praktis. Yang pertama melibatkan pencarian algoritma yang mampu belajar mengenali digit tulisan tangan dan memaksimalkan kinerja prediktifnya. Yang kedua sama sekali tidak melibatkan hal tersebut menunjukkan bahwa perolehan, pembersihan, dan representasi data setidaknya sama pentingnya dengan rekayasa algoritma.

Studi Kasus: Pengenalan Digit Tulis Tangan

Pengenalan digit tulisan tangan merupakan masalah penting dalam banyak aplikasi, termasuk penyortiran surat otomatis berdasarkan kode pos, pembacaan cek dan SPT otomatis, dan entri data untuk komputer genggam. Ini adalah area di mana kemajuan pesat telah dicapai, sebagian karena algoritma pembelajaran yang lebih baik dan sebagian karena ketersediaan set pelatihan yang lebih baik. Institut Sains dan Teknologi Nasional Amerika Serikat (NIST) telah mengarsipkan basis data berisi 60.000 digit berlabel, masing-masing berukuran $20 \times 20 = 400$ piksel dengan nilai skala abu-abu 8-bit. Ini telah menjadi salah satu masalah tolok ukur standar untuk membandingkan algoritma pembelajaran baru. Beberapa contoh digit ditunjukkan pada Gambar 9.36.



Gambar 9.36 Contoh digit tulisan tangan dari basis data NIST. Baris atas: contoh digit 0–9 yang mudah diidentifikasi. Baris bawah: contoh digit yang sama yang lebih sulit.

Berbagai pendekatan pembelajaran telah dicoba. Salah satu yang pertama, dan mungkin yang paling sederhana, adalah pengklasifikasi 3-tetangga-terdekat, yang juga memiliki keuntungan karena tidak memerlukan waktu pelatihan. Namun, sebagai algoritma berbasis memori, algoritma ini harus menyimpan seluruh 60.000 gambar, dan kinerja waktu prosesnya lambat. Tingkat kesalahan pengujiannya mencapai 2,4%.

Sebuah jaringan saraf tiruan satu lapis tersembunyi dirancang untuk permasalahan ini dengan 400 unit masukan (satu per piksel) dan 10 unit keluaran (satu per kelas). Dengan menggunakan validasi silang, ditemukan bahwa sekitar 300 unit tersembunyi memberikan kinerja terbaik. Dengan interkoneksi penuh antar lapisan, terdapat total 123.300 bobot. Jaringan ini mencapai tingkat kesalahan 1,6%. Serangkaian jaringan saraf tiruan khusus yang disebut LeNet dirancang untuk memanfaatkan struktur permasalahan bahwa masukan terdiri dari piksel dalam larik dua dimensi, dan bahwa perubahan kecil pada posisi atau kemiringan gambar tidaklah penting. Setiap jaringan memiliki lapisan input 32×32 unit, di mana 20×20 piksel dipusatkan sehingga setiap unit input disajikan dengan lingkungan piksel lokal. Ini diikuti oleh tiga lapisan unit tersembunyi.

Setiap lapisan terdiri dari beberapa bidang array $n \times n$, di mana n lebih kecil dari lapisan sebelumnya sehingga jaringan melakukan down-sampling input, dan di mana bobot setiap unit dalam bidang dibatasi menjadi identik, sehingga bidang bertindak sebagai detektor fitur: ia dapat memilih fitur seperti garis vertikal panjang atau busur setengah lingkaran pendek. Lapisan output memiliki 10 unit. Banyak versi arsitektur ini dicoba; salah satu yang representatif memiliki lapisan tersembunyi dengan 768, 192, dan 30 unit, masing-masing. Set pelatihan ditambah dengan menerapkan transformasi affine ke input aktual: menggeser, sedikit memutar, dan menskalakan gambar. (Tentu saja, transformasinya harus kecil, atau angka 6 akan berubah menjadi angka 9!) Tingkat kesalahan terbaik yang dicapai oleh LeNet adalah 0,9%.

Jaringan saraf tiruan yang ditingkatkan menggabungkan tiga salinan arsitektur LeNet, dengan salinan kedua dilatih pada campuran pola yang salah 50% dari salinan pertama, dan salinan ketiga dilatih pada pola yang tidak disetujui oleh kedua salinan pertama. Selama pengujian, ketiga jaringan tersebut memberikan suara dengan suara mayoritas. Tingkat kesalahan pengujian adalah 0,7%. Mesin vektor pendukung dengan 25.000 vektor pendukung mencapai tingkat kesalahan 1,1%. Hal ini luar biasa karena teknik SVM, seperti pendekatan tetangga terdekat yang sederhana, hampir tidak memerlukan pemikiran atau eksperimen berulang dari pihak pengembang, namun tetap mendekati kinerja LeNet, yang telah dikembangkan selama bertahun-tahun. Memang, mesin vektor pendukung tidak memanfaatkan struktur masalah, dan akan berkinerja sama baiknya jika piksel disajikan dalam urutan permutasi.

Mesin vektor pendukung virtual (VVM) dimulai dengan SVM biasa, lalu menyempurnakannya dengan teknik yang dirancang untuk memanfaatkan struktur

permasalahan. Alih-alih memungkinkan perkalian semua pasangan piksel, pendekatan ini berfokus pada kernel yang terbentuk dari pasangan piksel terdekat. Pendekatan ini juga melengkapi set pelatihan dengan transformasi contoh, seperti yang dilakukan LeNet. SVM virtual mencapai tingkat kesalahan terbaik yang tercatat hingga saat ini, yaitu 0,56%. Pencocokan bentuk (*shape matching*) adalah teknik dari visi komputer yang digunakan untuk menyelaraskan bagian-bagian yang bersesuaian dari dua gambar objek yang berbeda. Idennya adalah memilih satu set titik di masing-masing gambar, lalu menghitung, untuk setiap titik di gambar pertama, titik mana di gambar kedua yang bersesuaian dengannya. Dari penyelarasan ini, kami kemudian menghitung transformasi antar gambar. Transformasi ini memberi kami ukuran jarak antar gambar.

Ukuran jarak ini lebih termotivasi daripada hanya menghitung jumlah piksel yang berbeda, dan ternyata algoritma 3 tetangga terdekat yang menggunakan ukuran jarak ini berkinerja sangat baik. Dengan pelatihan hanya pada 20.000 dari 60.000 digit, dan menggunakan 100 titik sampel per gambar yang diekstraksi dari detektor tepi Canny, pengklasifikasi pencocokan bentuk mencapai galat uji 0,63%. Manusia diperkirakan memiliki tingkat galat sekitar 0,2% dalam soal ini. Angka ini agak meragukan karena manusia belum diuji secara ekstensif seperti algoritma pembelajaran mesin. Pada kumpulan data digit serupa dari Layanan Pos Amerika Serikat, galat manusia mencapai 2,5%.

Gambar berikut merangkum tingkat galat, kinerja waktu proses, kebutuhan memori, dan jumlah waktu pelatihan untuk tujuh algoritma yang telah kita bahas. Gambar ini juga menambahkan ukuran lain, yaitu persentase digit yang harus ditolak untuk mencapai galat 0,5%. Misalnya, jika SVM diizinkan untuk menolak 1,8% masukan yaitu, meneruskannya kepada orang lain untuk membuat penilaian akhir maka tingkat kesalahannya pada 98,2% masukan yang tersisa berkurang dari 1,1% menjadi 0,5%. Tabel berikut merangkum tingkat kesalahan dan beberapa karakteristik lain dari ketujuh teknik yang telah kita bahas.

Metode	3-NN	300 Hidden	LeNet	LeNet + Boosting	SVM	SVM Virtual	Shape Match
Tingkat galat (dalam %)	2.4	1.6	0.9	0.7	1.1	0.56	0.63
Waktu eksekusi (milidetik/digit)	1000	10	30	50	2000	200	-
Kebutuhan memori (MB)	12	0.49	0.012	0.21	11	-	-
Waktu pelatihan (hari)	0	7	14	30	10	-	-
% Data ditolak untuk mencapai galat 0.5%	8.1	3.2	1.8	0.5	1.8	-	-

Studi Kasus: Makna Kata dan Harga Rumah

Dalam buku teks, kita perlu menangani data mainan sederhana untuk menyampaikan gagasan: sekumpulan data kecil, biasanya dalam dua dimensi. Namun dalam aplikasi praktis pembelajaran mesin, kumpulan data tersebut biasanya besar, multidimensi, dan berantakan. Data tidak diserahkan kepada analis dalam bentuk paket nilai (x, y) yang telah dikemas sebelumnya; melainkan, analis perlu keluar dan memperoleh data yang tepat. Ada tugas yang

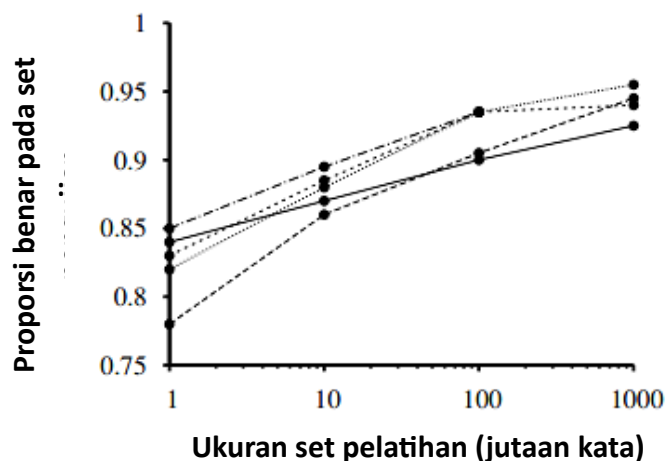
harus diselesaikan, dan sebagian besar masalah rekayasa adalah memutuskan data apa yang diperlukan untuk menyelesaikan tugas tersebut; bagian yang lebih kecil adalah memilih dan menerapkan metode pembelajaran mesin yang tepat untuk memproses data.

Gambar 9.37 menunjukkan contoh dunia nyata yang umum, membandingkan lima algoritma pembelajaran pada tugas klasifikasi makna kata (dengan kalimat seperti "Bank bangkrut," klasifikasikan kata "bank" sebagai "bank uang" atau "tepi sungai"). Intinya adalah para peneliti pembelajaran mesin berfokus terutama pada arah vertikal: Bisakah saya menciptakan algoritma pembelajaran baru yang berkinerja lebih baik daripada algoritma yang telah dipublikasikan sebelumnya pada set pelatihan standar 1 juta kata? Namun, grafik menunjukkan masih ada ruang untuk perbaikan pada arah horizontal: alih-alih menciptakan algoritma baru, yang perlu saya lakukan hanyalah mengumpulkan 10 juta kata data pelatihan; bahkan algoritma terburuk dengan 10 juta kata berkinerja lebih baik daripada algoritma terbaik dengan 1 juta kata. Seiring kita mengumpulkan lebih banyak data, kurvanya terus meningkat, sehingga perbedaan antar algoritma menjadi lebih kecil.

Pertimbangkan masalah lain: tugas memperkirakan nilai sebenarnya dari rumah yang dijual. Pada Gambar 9.13, kami menunjukkan versi simulasi dari masalah ini, dengan melakukan regresi linier dari ukuran rumah terhadap harga yang diminta. Anda mungkin menyadari banyak keterbatasan model ini. Pertama, model ini mengukur hal yang salah: kita ingin memperkirakan harga jual rumah, bukan harga yang diminta. Untuk menyelesaikan tugas ini, kita membutuhkan data penjualan aktual. Namun, bukan berarti kita harus membuang data harga yang diminta kita bisa menggunakannya sebagai salah satu fitur input. Selain ukuran rumah, kita juga membutuhkan informasi lebih lanjut: jumlah kamar, kamar tidur, dan kamar mandi; apakah dapur dan kamar mandi baru saja direnovasi; usia rumah; kita juga membutuhkan informasi tentang tanah dan lingkungannya.

Namun, bagaimana kita mendefinisikan lingkungan? Berdasarkan kode pos? Bagaimana jika sebagian dari satu kode pos berada di sisi "salah" jalan raya atau rel kereta api, sementara sebagian lainnya diinginkan? Bagaimana dengan distrik sekolah? Haruskah nama distrik sekolah menjadi fitur, atau nilai ujian rata-rata? Selain memutuskan fitur apa yang akan disertakan, kita juga harus berurusan dengan data yang hilang; setiap daerah memiliki kebiasaan yang berbeda tentang data yang dilaporkan, dan setiap kasus akan selalu kehilangan beberapa data.

Jika data yang Anda inginkan tidak tersedia, mungkin Anda dapat membuat situs jejaring sosial untuk mendorong orang-orang berbagi dan mengoreksi data. Pada akhirnya, proses memutuskan fitur apa yang akan digunakan, dan bagaimana menggunakannya, sama pentingnya dengan memilih antara regresi linier, pohon keputusan, atau bentuk pembelajaran lainnya.



Gambar 9.37 Kurva pembelajaran untuk lima algoritma pembelajaran pada tugas yang sama.

Perhatikan bahwa tampaknya terdapat lebih banyak ruang untuk perbaikan pada arah horizontal (data pelatihan yang lebih banyak) daripada arah vertikal (algoritma pembelajaran mesin yang berbeda).

Meskipun demikian, seseorang memang harus memilih satu (atau beberapa) metode untuk suatu masalah. Tidak ada cara pasti untuk memilih metode terbaik, tetapi ada beberapa panduan kasar. Pohon keputusan bagus ketika terdapat banyak fitur diskrit dan Anda yakin bahwa banyak di antaranya mungkin tidak relevan. Metode nonparametrik bagus ketika Anda memiliki banyak data dan tanpa pengetahuan sebelumnya, dan ketika Anda tidak ingin terlalu khawatir tentang memilih fitur yang tepat (asalkan jumlahnya kurang dari 20). Namun, metode nonparametrik biasanya memberi Anda fungsi h yang lebih mahal untuk dijalankan. Mesin vektor pendukung sering dianggap sebagai metode terbaik untuk dicoba terlebih dahulu, asalkan kumpulan datanya tidak terlalu besar.

9.12 KESIMPULAN

Bab ini berfokus pada pembelajaran induktif fungsi dari contoh. Poin-poin utamanya adalah sebagai berikut:

- Pembelajaran memiliki banyak bentuk, bergantung pada sifat agen, komponen yang akan ditingkatkan, dan umpan balik yang tersedia.
- Jika umpan balik yang tersedia memberikan jawaban yang benar untuk masukan contoh, maka permasalahan pembelajaran tersebut disebut pembelajaran terbimbing. Tugasnya adalah mempelajari fungsi $y = h(x)$. Mempelajari fungsi bernilai diskrit disebut klasifikasi; mempelajari fungsi kontinu disebut regresi.
- Pembelajaran induktif melibatkan pencarian hipotesis yang sesuai dengan contoh. Pisau cukur Ockham menyarankan untuk memilih hipotesis konsisten yang paling sederhana. Kesulitan tugas ini bergantung pada representasi yang dipilih.
- Pohon keputusan dapat merepresentasikan semua fungsi Boolean. Heuristik perolehan informasi menyediakan metode yang efisien untuk menemukan pohon keputusan yang sederhana dan konsisten.

- Performa algoritma pembelajaran diukur dengan kurva pembelajaran, yang menunjukkan akurasi prediksi pada set uji sebagai fungsi dari ukuran set pelatihan. • Ketika terdapat beberapa model yang dapat dipilih, validasi silang dapat digunakan untuk memilih model yang dapat digeneralisasi dengan baik.
- Terkadang tidak semua kesalahan sama. Fungsi kerugian memberi tahu kita seberapa parah setiap kesalahan; tujuannya adalah meminimalkan kerugian atas set validasi.
- Teori pembelajaran komputasional menganalisis kompleksitas sampel dan kompleksitas komputasional dari pembelajaran induktif. Terdapat trade-off antara ekspresivitas bahasa hipotesis dan kemudahan pembelajaran.
- Regresi linier adalah model yang banyak digunakan. Parameter optimal model regresi linier dapat ditemukan melalui pencarian penurunan gradien, atau dihitung secara eksak.
- Pengklasifikasi linier dengan ambang batas keras—juga dikenal sebagai perceptron—dapat dilatih dengan aturan pembaruan bobot sederhana agar sesuai dengan data yang dapat dipisahkan secara linier. Dalam kasus lain, aturan tersebut gagal konvergen.
- Regresi logistik menggantikan ambang batas keras perceptron dengan ambang batas lunak yang didefinisikan oleh fungsi logistik. Penurunan gradien berfungsi dengan baik bahkan untuk data bising yang tidak dapat dipisahkan secara linear.
- Jaringan saraf tiruan merepresentasikan fungsi nonlinier kompleks dengan jaringan unit ambang batas linear. Jaringan saraf tiruan umpan-maju multilapis dapat merepresentasikan fungsi apa pun, dengan jumlah unit yang cukup. Algoritma propagasi balik mengimplementasikan penurunan gradien dalam ruang parameter untuk meminimalkan kesalahan keluaran.
- Model nonparametrik menggunakan semua data untuk membuat setiap prediksi, alih-alih mencoba meringkas data terlebih dahulu dengan beberapa parameter. Contohnya termasuk tetangga terdekat dan regresi terbobot lokal.
- Mesin vektor pendukung menemukan pemisah linear dengan margin maksimum untuk meningkatkan kinerja generalisasi pengklasifikasi. Metode kernel secara implisit mengubah data masukan menjadi ruang berdimensi tinggi di mana pemisah linear mungkin ada, meskipun data asli tidak dapat dipisahkan.
- Metode ansambel seperti boosting seringkali berkinerja lebih baik daripada metode individual. Dalam pembelajaran daring, kita dapat menggabungkan pendapat para pakar untuk mendekati kinerja pakar terbaik, bahkan ketika distribusi data terus berubah.

DAFTAR PUSTAKA

- Alghanemi, J., & Al Mubarak, M. (2022). The role of artificial intelligence in knowledge management. In *Future of Organizations and Work After the 4th Industrial Revolution: The Role of Artificial Intelligence, Big Data, Automation, and Robotics* (pp. 359-373). Cham: Springer International Publishing.
- Al-Sharafi, M. A., Al-Emran, M., Iranmanesh, M., Al-Qaysi, N., Iahad, N. A., & Arpaci, I. (2023). Understanding the impact of knowledge management factors on the sustainable use of AI-based chatbots for educational purposes using a hybrid SEM-ANN approach. *Interactive Learning Environments*, 31(10), 7491-7510.
- Bag, S., Gupta, S., Kumar, A., & Sivarajah, U. (2021). An integrated artificial intelligence framework for knowledge creation and B2B marketing rational decision making for improving firm performance. *Industrial marketing management*, 92, 178-189.
- Chen, J., & Nonaka, I. (Eds.). (2022). *The Routledge companion to knowledge management*. Taylor & Francis.
- Chen, Z. (2023). Artificial intelligence-virtual trainer: Innovative didactics aimed at personalized training needs. *Journal of the Knowledge Economy*, 14(2), 2007-2025.
- Cheng, C. H., Li, M. H., Tang, B. J., & Cheng, Y. R. (2024). The impact of knowledge management and organizational learning promotion in small and medium enterprises on the implementation of industry 4.0 and competitiveness. *Administrative Sciences*, 14(8), 161.
- De Bem Machado, A., Secinaro, S., Calandra, D., & Lanza Longa, F. (2022). Knowledge management and digital transformation for Industry 4.0: a structured literature review. *Knowledge Management Research & Practice*, 20(2), 320-338.
- Galgotia, D., & Lakshmi, N. (2024). RETRACTED ARTICLE: Development of IOT-based methodology for the execution of knowledge management using artificial intelligence in higher education system. *Soft Computing*, 28(Suppl 2), 599-599.
- Huang, Z., He, J., & Ren, X. (2024). RETRACTED ARTICLE: Application of artificial intelligence in enterprise knowledge management performance evaluation. *Knowledge Management Research & Practice*, 22(6), LXXIII-LXXXI.
- Iaia, L., Nespoli, C., Vicentini, F., Pironti, M., & Genovino, C. (2024). Supporting the implementation of AI in business communication: the role of knowledge management. *Journal of Knowledge Management*, 28(1), 85-95.
- Idrus, S., Jannah, K. D., Wicaksono, M. B. A., Tanjung, S. P., & Amin, F. (2023). Digital transformation and artificial intelligence in marketing for startups using a customer knowledge management approach. *International Journal of Artificial Intelligence Research*, 6(1.1).

- Jarrahi, M. H., Askay, D., Eshraghi, A., & Smith, P. (2023). Artificial intelligence and knowledge management: A partnership between human and AI. *Business Horizons*, 66(1), 87-99.
- Khan, A. N., Mehmood, K., & Soomro, M. A. (2024). Knowledge management-based artificial intelligence (AI) adoption in construction SMEs: the moderating role of knowledge integration. *IEEE transactions on engineering management*, 71, 10874-10884.
- Kitsios, F., & Kamariotou, M. (2021). Artificial intelligence and business strategy towards digital transformation: A research agenda. *Sustainability*, 13(4), 2025.
- Majumder, S., & Dey, N. (2022). *AI-empowered knowledge management*. Springer.
- Malik, A., De Silva, M. T., Budhwar, P., & Srikanth, N. R. (2021). Elevating talents' experience through innovative artificial intelligence-mediated knowledge sharing: Evidence from an IT-multinational enterprise. *Journal of International Management*, 27(4), 100871.
- Malik, N., Tripathi, S. N., Kar, A. K., & Gupta, S. (2022). Impact of artificial intelligence on employees working in industry 4.0 led organizations. *International Journal of Manpower*, 43(2), 334-354.
- Nakash, M., & Bouhnik, D. (2021). Knowledge management is not dead. It has changed its appearance. And it will continue to change. *Knowledge and Process Management*, 28(1), 29-39.
- Nguyen, T. M., & Malik, A. (2022). Impact of knowledge sharing on employees' service quality: the moderating role of artificial intelligence. *International Marketing Review*, 39(3), 482-508.
- Odugbesan, J. A., Aghazadeh, S., Al Qaralleh, R. E., & Sogeke, O. S. (2023). Green talent management and employees' innovative work behavior: the roles of artificial intelligence and transformational leadership. *Journal of knowledge management*, 27(3), 696-716.
- Olan, F., Arakpogun, E. O., Suklan, J., Nakpodia, F., Damij, N., & Jayawickrama, U. (2022). Artificial intelligence and knowledge sharing: Contributing factors to organizational performance. *Journal of Business Research*, 145, 605-615.
- Papagiannidis, E., Mikalef, P., Krogstie, J., & Conboy, K. (2022, September). From responsible AI governance to competitive performance: the mediating role of knowledge management capabilities. In *Conference on e-Business, e-Services and e-Society* (pp. 58-69). Cham: Springer International Publishing.
- Razia, B., Awwad, B., & Taqi, N. (2023). The relationship between artificial intelligence (AI) and its aspects in higher education. *Development and learning in organizations: an international journal*, 37(3), 21-23.
- Song, Y., & Wu, R. (2021). Analysing human-computer interaction behaviour in human resource management system based on artificial intelligence technology. *Knowledge Management Research & Practice*, 1-10.

- Sundaresan, S., & Zhang, Z. (2022). AI-enabled knowledge sharing and learning: redesigning roles and processes. *International journal of organizational analysis*, 30(4), 983-999.
- Taherdoost, H., & Madanchian, M. (2023). Artificial intelligence and knowledge management: Impacts, benefits, and implementation. *Computers*, 12(4), 72.
- Thakral, P., Sharma, D., & Ghosh, K. (2024). Evidence-based knowledge management: a topic modeling analysis of research on knowledge management and analytics. *VINE Journal of Information and Knowledge Management Systems*.
- Tomé, E., Gromova, E., & Hatch, A. (2022). Knowledge management and COVID-19: Technology, people and processes. *Knowledge and Process Management*, 29(1), 70-78.
- Vadari, S., & Desik, P. A. (2021). The role of AI/ML in enhancing knowledge management systems. *IUP Journal of Knowledge Management*, 19(2), 7-31.
- Venkitachalam, K., & Schiuma, G. (2022). Strategic knowledge management (SKM) in the digital age—insights and possible research directions. *Journal of Strategy and Management*, 15(2), 169-174.
- Wang, S., & Zhang, H. (2024). Green entrepreneurship success in the age of generative artificial intelligence: The interplay of technology adoption, knowledge management, and government support. *Technology in Society*, 79, 102744.
- Xu, M., Liu, D., & Zhang, Y. (2022). Design of interactive teaching system of physical training based on artificial intelligence. *Journal of Information & Knowledge Management*, 21(Supp02), 2240021.

Manajemen Pengetahuan Kecerdasan Buatan (AI)

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

BIO DATA PENULIS



Penulis memiliki berbagai disiplin ilmu yang diperoleh dari Universitas Diponegoro (UNDIP) Semarang. dan dari Universitas Kristen Satya Wacana (UKSW) Salatiga. Disiplin ilmu itu antara lain teknik elektro, komputer, manajemen dan ilmu sosiologi. Penulis memiliki pengalaman kerja pada industri elektronik dan sertifikasi keahlian dalam bidang Jaringan Internet, Telekomunikasi, Artificial Intelligence, Internet Of Things (IoT), Augmented Reality (AR), Technopreneurship, Internet Marketing dan bidang pengolahan dan analisa data (komputer statistik).

Penulis adalah pendiri dari Universitas Sains dan Teknologi Komputer (Universitas STEKOM) dan juga seorang dosen yang memiliki Jabatan Fungsional Akademik Lektor Kepala (Associate Professor) yang telah menghasilkan puluhan Buku Ajar ber ISBN, HAKI dari beberapa karya cipta dan Hak Paten pada produk IPTEK. Sejak tahun 2023 penulis tercatat sebagai Dosen luar biasa di Fakultas Ekonomi & Bisnis (FEB) Universitas Diponegoro Semarang. Penulis juga terlibat dalam berbagai organisasi profesi dan industri yang terkait dengan dunia usaha dan industri, khususnya dalam pengembangan sumber daya manusia yang unggul untuk memenuhi kebutuhan dunia kerja secara nyata.



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-634-7227-29-4 (PDF)



9

786347

227294