13 QUANTIFYING UNCERTAINTY

In which we see how an agent can tame uncertainty with degrees of belief.

13.1 ACTING UNDER UNCERTAINTY

UNCERTAINTY

Agents may need to handle **uncertainty**, whether due to partial observability, nondeterminism, or a combination of the two. An agent may never know for certain what state it's in or where it will end up after a sequence of actions.

We have seen problem-solving agents (Chapter 4) and logical agents (Chapters 7 and 11) designed to handle uncertainty by keeping track of a **belief state**—a representation of the set of all possible world states that it might be in—and generating a contingency plan that handles every possible eventuality that its sensors may report during execution. Despite its many virtues, however, this approach has significant drawbacks when taken literally as a recipe for creating agent programs:

- When interpreting partial sensor information, a logical agent must consider *every logically possible* explanation for the observations, no matter how unlikely. This leads to impossible large and complex belief-state representations.
- A correct contingent plan that handles every eventuality can grow arbitrarily large and must consider arbitrarily unlikely contingencies.
- Sometimes there is no plan that is guaranteed to achieve the goal—yet the agent must act. It must have some way to compare the merits of plans that are not guaranteed.

Suppose, for example, that an automated taxi!automated has the goal of delivering a passenger to the airport on time. The agent forms a plan, A_{90}, that involves leaving home 90 minutes before the flight departs and driving at a reasonable speed. Even though the airport is only about 5 miles away, a logical taxi agent will not be able to conclude with certainty that "Plan A_{90} will get us to the airport in time." Instead, it reaches the weaker conclusion "Plan A_{90} will get us to the airport in time, as long as the car doesn't break down or run out of gas, and I don't get into an accident, and there are no accidents on the bridge, and the plane doesn't leave early, and no meteorite hits the car, and . . . ." None of these conditions can be

deduced for sure, so the plan's success cannot be inferred. This is the **qualification problem** (page 268), for which we so far have seen no real solution.

Nonetheless, in some sense A_{90} *is* in fact the right thing to do. What do we mean by this? As we discussed in Chapter 2, we mean that out of all the plans that could be executed, A_{90} is expected to maximize the agent's performance measure (where the expectation is relative to the agent's knowledge about the environment). The performance measure includes getting to the airport in time for the flight, avoiding a long, unproductive wait at the airport, and avoiding speeding tickets along the way. The agent's knowledge cannot guarantee any of these outcomes for A_{90}, but it can provide some degree of belief that they will be achieved. Other plans, such as A_{180}, might increase the agent's belief that it will get to the airport on time, but also increase the likelihood of a long wait. *The right thing to do—the **rational decision**—therefore depends on both the relative importance of various goals and the likelihood that, and degree to which, they will be achieved.* The remainder of this section hones these ideas, in preparation for the development of the general theories of uncertain reasoning and rational decisions that we present in this and subsequent chapters.

13.1.1 Summarizing uncertainty

Let's consider an example of uncertain reasoning: diagnosing a dental patient's toothache. Diagnosis—whether for medicine, automobile repair, or whatever—almost always involves uncertainty. Let us try to write rules for dental diagnosis using propositional logic, so that we can see how the logical approach breaks down. Consider the following simple rule:

$$Toothache \Rightarrow Cavity\ .$$

The problem is that this rule is wrong. Not all patients with toothaches have cavities; some of them have gum disease, an abscess, or one of several other problems:

$$Toothache \Rightarrow Cavity \vee GumProblem \vee Abscess \ldots$$

Unfortunately, in order to make the rule true, we have to add an almost unlimited list of possible problems. We could try turning the rule into a causal rule:

$$Cavity \Rightarrow Toothache\ .$$

But this rule is not right either; not all cavities cause pain. The only way to fix the rule is to make it logically exhaustive: to augment the left-hand side with all the qualifications required for a cavity to cause a toothache. Trying to use logic to cope with a domain like medical diagnosis thus fails for three main reasons:

LAZINESS
- **Laziness**: It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules.

THEORETICAL IGNORANCE
- **Theoretical ignorance**: Medical science has no complete theory for the domain.

PRACTICAL IGNORANCE
- **Practical ignorance**: Even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.

The connection between toothaches and cavities is just not a logical consequence in either direction. This is typical of the medical domain, as well as most other judgmental domains: law, business, design, automobile repair, gardening, dating, and so on. The agent's knowledge

DEGREE OF BELIEF
PROBABILITY THEORY

can at best provide only a **degree of belief** in the relevant sentences. Our main tool for dealing with degrees of belief is **probability theory**. In the terminology of Section 8.1, the **ontological commitments** of logic and probability theory are the same—that the world is composed of facts that do or do not hold in any particular case—but the **epistemological commitments** are different: a logical agent believes each sentence to be true or false or has no opinion, whereas a probabilistic agent may have a numerical degree of belief between 0 (for sentences that are certainly false) and 1 (certainly true).

Probability provides a way of **summarizing** *the uncertainty that comes from our laziness and ignorance,* thereby solving the qualification problem. We might not know for sure what afflicts a particular patient, but we believe that there is, say, an 80% chance—that is, a probability of 0.8—that the patient who has a toothache has a cavity. That is, we expect that out of all the situations that are indistinguishable from the current situation as far as our knowledge goes, the patient will have a cavity in 80% of them. This belief could be derived from statistical data—80% of the toothache patients seen so far have had cavities—or from some general dental knowledge, or from a combination of evidence sources.

One confusing point is that at the time of our diagnosis, there is no uncertainty in the actual world: the patient either has a cavity or doesn't. So what does it mean to say the probability of a cavity is 0.8? Shouldn't it be either 0 or 1? The answer is that probability statements are made with respect to a knowledge state, not with respect to the real world. We say "The probability that the patient has a cavity, *given that she has a toothache*, is 0.8." If we later learn that the patient has a history of gum disease, we can make a different statement: "The probability that the patient has a cavity, given that she has a toothache and a history of gum disease, is 0.4." If we gather further conclusive evidence against a cavity, we can say "The probability that the patient has a cavity, given all we now know, is almost 0." Note that these statements do not contradict each other; each is a separate assertion about a different knowledge state.

13.1.2 Uncertainty and rational decisions

Consider again the A_{90} plan for getting to the airport. Suppose it gives us a 97% chance of catching our flight. Does this mean it is a rational choice? Not necessarily: there might be other plans, such as A_{180}, with higher probabilities. If it is vital not to miss the flight, then it is worth risking the longer wait at the airport. What about A_{1440}, a plan that involves leaving home 24 hours in advance? In most circumstances, this is not a good choice, because although it almost guarantees getting there on time, it involves an intolerable wait—not to mention a possibly unpleasant diet of airport food.

PREFERENCE
OUTCOME

To make such choices, an agent must first have **preferences** between the different possible **outcomes** of the various plans. An outcome is a completely specified state, including such factors as whether the agent arrives on time and the length of the wait at the airport. We

UTILITY THEORY

use **utility theory** to represent and reason with preferences. (The term **utility** is used here in the sense of "the quality of being useful," not in the sense of the electric company or water works.) Utility theory says that every state has a degree of usefulness, or utility, to an agent and that the agent will prefer states with higher utility.

The utility of a state is relative to an agent. For example, the utility of a state in which White has checkmated Black in a game of chess is obviously high for the agent playing White, but low for the agent playing Black. But we can't go strictly by the scores of 1, 1/2, and 0 that are dictated by the rules of tournament chess—some players (including the authors) might be thrilled with a draw against the world champion, whereas other players (including the former world champion) might not. There is no accounting for taste or preferences: you might think that an agent who prefers jalapeño bubble-gum ice cream to chocolate chocolate chip is odd or even misguided, but you could not say the agent is irrational. A utility function can account for any set of preferences—quirky or typical, noble or perverse. Note that utilities can account for altruism, simply by including the welfare of others as one of the factors.

DECISION THEORY

Preferences, as expressed by utilities, are combined with probabilities in the general theory of rational decisions called **decision theory**:

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

MAXIMUM EXPECTED UTILITY

The fundamental idea of decision theory is that *an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action.* This is called the principle of **maximum expected utility** (MEU). Note that "expected" might seem like a vague, hypothetical term, but as it is used here it has a precise meaning: it means the "average," or "statistical mean" of the outcomes, weighted by the probability of the outcome. We saw this principle in action in Chapter 5 when we touched briefly on optimal decisions in backgammon; it is in fact a completely general principle.

Figure 13.1 sketches the structure of an agent that uses decision theory to select actions. The agent is identical, at an abstract level, to the agents described in Chapters 4 and 7 that maintain a belief state reflecting the history of percepts to date. The primary difference is that the decision-theoretic agent's belief state represents not just the *possibilities* for world states but also their *probabilities*. Given the belief state, the agent can make probabilistic predictions of action outcomes and hence select the action with highest expected utility. This chapter and the next concentrate on the task of representing and computing with probabilistic information in general. Chapter 15 deals with methods for the specific tasks of representing and updating the belief state over time and predicting the environment. Chapter 16 covers utility theory in more depth, and Chapter 17 develops algorithms for planning sequences of actions in uncertain environments.

13.2 Basic Probability Notation

For our agent to represent and use probabilistic information, we need a formal language. The language of probability theory has traditionally been informal, written by human mathematicians to other human mathematicians. Appendix A includes a standard introduction to elementary probability theory; here, we take an approach more suited to the needs of AI and more consistent with the concepts of formal logic.

```
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
```

Figure 13.1 A decision-theoretic agent that selects rational actions.

13.2.1 What probabilities are about

Like logical assertions, probabilistic assertions are about possible worlds. Whereas logical assertions say which possible worlds are strictly ruled out (all those in which the assertion is false), probabilistic assertions talk about how probable the various worlds are. In probability theory, the set of all possible worlds is called the **sample space**. The possible worlds are *mutually exclusive* and *exhaustive*—two possible worlds cannot both be the case, and one possible world must be the case. For example, if we are about to roll two (distinguishable) dice, there are 36 possible worlds to consider: (1,1), (1,2), ..., (6,6). The Greek letter Ω (uppercase omega) is used to refer to the sample space, and ω (lowercase omega) refers to elements of the space, that is, particular possible worlds.

SAMPLE SPACE

PROBABILITY MODEL

A fully specified **probability model** associates a numerical probability $P(\omega)$ with each possible world.[1] The basic axioms of probability theory say that every possible world has a probability between 0 and 1 and that the total probability of the set of possible worlds is 1:

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

For example, if we assume that each die is fair and the rolls don't interfere with each other, then each of the possible worlds (1,1), (1,2), ..., (6,6) has probability 1/36. On the other hand, if the dice conspire to produce the same number, then the worlds (1,1), (2,2), (3,3), etc., might have higher probabilities, leaving the others with lower probabilities.

Probabilistic assertions and queries are not usually about particular possible worlds, but about sets of them. For example, we might be interested in the cases where the two dice add up to 11, the cases where doubles are rolled, and so on. In probability theory, these sets are called **events**—a term already used extensively in Chapter 12 for a different concept. In AI, the sets are always described by **propositions** in a formal language. (One such language is described in Section 13.2.2.) For each proposition, the corresponding set contains just those possible worlds in which the proposition holds. The probability associated with a proposition

EVENT

[1] For now, we assume a discrete, countable set of worlds. The proper treatment of the continuous case brings in certain complications that are less relevant for most purposes in AI.

is defined to be the sum of the probabilities of the worlds in which it holds:

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

For example, when rolling fair dice, we have $P(\mathit{Total} = 11) = P((5,6)) + P((6,5)) = 1/36 + 1/36 = 1/18$. Note that probability theory does not require complete knowledge of the probabilities of each possible world. For example, if we believe the dice conspire to produce the same number, we might *assert* that $P(\mathit{doubles}) = 1/4$ without knowing whether the dice prefer double 6 to double 2. Just as with logical assertions, this assertion *constrains* the underlying probability model without fully determining it.

UNCONDITIONAL PROBABILITY
PRIOR PROBABILITY

Probabilities such as $P(\mathit{Total} = 11)$ and $P(\mathit{doubles})$ are called **unconditional** or **prior probabilities** (and sometimes just "priors" for short); they refer to degrees of belief in propositions *in the absence of any other information*. Most of the time, however, we have *some* information, usually called **evidence**, that has already been revealed. For example, the first die may already be showing a 5 and we are waiting with bated breath for the other one to stop spinning. In that case, we are interested not in the unconditional probability of rolling doubles, but the **conditional** or **posterior** probability (or just "posterior" for short) of rolling doubles *given that the first die is a 5*. This probability is written $P(\mathit{doubles} \mid \mathit{Die}_1 = 5)$, where the " $\mid$ " is pronounced "given." Similarly, if I am going to the dentist for a regular checkup, the probability $P(\mathit{cavity}) = 0.2$ might be of interest; but if I go to the dentist because I have a toothache, it's $P(\mathit{cavity} \mid \mathit{toothache}) = 0.6$ that matters. Note that the precedence of " $\mid$ " is such that any expression of the form $P(\ldots \mid \ldots)$ always means $P((\ldots)|(\ldots))$.

EVIDENCE

CONDITIONAL PROBABILITY
POSTERIOR PROBABILITY

It is important to understand that $P(\mathit{cavity}) = 0.2$ is still *valid* after $\mathit{toothache}$ is observed; it just isn't especially useful. When making decisions, an agent needs to condition on *all* the evidence it has observed. It is also important to understand the difference between conditioning and logical implication. The assertion that $P(\mathit{cavity} \mid \mathit{toothache}) = 0.6$ does not mean "Whenever $\mathit{toothache}$ is true, conclude that cavity is true with probability 0.6" rather it means "Whenever $\mathit{toothache}$ is true *and we have no further information*, conclude that cavity is true with probability 0.6." The extra condition is important; for example, if we had the further information that the dentist found no cavities, we definitely would not want to conclude that cavity is true with probability 0.6; instead we need to use $P(\mathit{cavity}|\mathit{toothache} \wedge \neg \mathit{cavity}) = 0$.

Mathematically speaking, conditional probabilities are defined in terms of unconditional probabilities as follows: for any propositions a and b, we have

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}\ , \tag{13.3}$$

which holds whenever $P(b) > 0$. For example,

$$P(\mathit{doubles} \mid \mathit{Die}_1 = 5) = \frac{P(\mathit{doubles} \wedge \mathit{Die}_1 = 5)}{P(\mathit{Die}_1 = 5)}\ .$$

The definition makes sense if you remember that observing b rules out all those possible worlds where b is false, leaving a set whose total probability is just $P(b)$. Within that set, the a-worlds satisfy $a \wedge b$ and constitute a fraction $P(a \wedge b)/P(b)$.

The definition of conditional probability, Equation (13.3), can be written in a different form called the **product rule**:

PRODUCT RULE

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

The product rule is perhaps easier to remember: it comes from the fact that, for a and b to be true, we need b to be true, and we also need a to be true given b.

13.2.2 The language of propositions in probability assertions

In this chapter and the next, propositions describing sets of possible worlds are written in a notation that combines elements of propositional logic and constraint satisfaction notation. In the terminology of Section 2.4.7, it is a **factored representation**, in which a possible world is represented by a set of variable/value pairs.

RANDOM VARIABLE

Variables in probability theory are called **random variables** and their names begin with an uppercase letter. Thus, in the dice example, $Total$ and Die_1 are random variables. Every

DOMAIN

random variable has a **domain**—the set of possible values it can take on. The domain of $Total$ for two dice is the set $\{2, \ldots, 12\}$ and the domain of Die_1 is $\{1, \ldots, 6\}$. A Boolean random variable has the domain $\{true, false\}$ (notice that values are always lowercase); for example, the proposition that doubles are rolled can be written as $Doubles = true$. By convention, propositions of the form $A = true$ are abbreviated simply as a, while $A = false$ is abbreviated as $\neg a$. (The uses of $doubles$, $cavity$, and $toothache$ in the preceding section are abbreviations of this kind.) As in CSPs, domains can be sets of arbitrary tokens; we might choose the domain of Age to be $\{juvenile, teen, adult\}$ and the domain of $Weather$ might be $\{sunny, rain, cloudy, snow\}$. When no ambiguity is possible, it is common to use a value by itself to stand for the proposition that a particular variable has that value; thus, $sunny$ can stand for $Weather = sunny$.

The preceding examples all have finite domains. Variables can have infinite domains, too—either discrete (like the integers) or continuous (like the reals). For any variable with an ordered domain, inequalities are also allowed, such as $NumberOfAtomsInUniverse \geq 10^{70}$.

Finally, we can combine these sorts of elementary propositions (including the abbreviated forms for Boolean variables) by using the connectives of propositional logic. For example, we can express "The probability that the patient has a cavity, given that she is a teenager with no toothache, is 0.1" as follows:

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

Sometimes we will want to talk about the probabilities of *all* the possible values of a random variable. We could write:

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

but as an abbreviation we will allow

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

where the bold **P** indicates that the result is a vector of numbers, and where we assume a predefined ordering $\langle sunny, rain, cloudy, snow\rangle$ on the domain of $Weather$. We say that the **P** statement defines a **probability distribution** for the random variable $Weather$. The **P** notation is also used for conditional distributions: $\mathbf{P}(X \mid Y)$ gives the values of $P(X = x_i \mid Y = y_j)$ for each possible i, j pair.

PROBABILITY DISTRIBUTION

For continuous variables, it is not possible to write out the entire distribution as a vector, because there are infinitely many values. Instead, we can define the probability that a random variable takes on some value x as a parameterized function of x. For example, the sentence

$$P(NoonTemp = x) = Uniform_{[18C,26C]}(x)$$

expresses the belief that the temperature at noon is distributed uniformly between 18 and 26 degrees Celsius. We call this a **probability density function**.

PROBABILITY DENSITY FUNCTION

Probability density functions (sometimes called **pdfs**) differ in meaning from discrete distributions. Saying that the probability density is uniform from $18C$ to $26C$ means that there is a 100% chance that the temperature will fall somewhere in that $8C$-wide region and a 50% chance that it will fall in any $4C$-wide region, and so on. We write the probability density for a continuous random variable X at value x as $P(X = x)$ or just $P(x)$; the intuitive definition of $P(x)$ is the probability that X falls within an arbitrarily small region beginning at x, divided by the width of the region:

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

For $NoonTemp$ we have

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

where C stands for centigrade (not for a constant). In $P(NoonTemp = 20.18C) = \frac{1}{8C}$, note that $\frac{1}{8C}$ is not a probability, it is a probability density. The probability that $NoonTemp$ is *exactly* $20.18C$ is zero, because $20.18C$ is a region of width 0. Some authors use different symbols for discrete distributions and density functions; we use P in both cases, since confusion seldom arises and the equations are usually identical. Note that probabilities are unitless numbers, whereas density functions are measured with a unit, in this case reciprocal degrees.

In addition to distributions on single variables, we need notation for distributions on multiple variables. Commas are used for this. For example, $\mathbf{P}(Weather, Cavity)$ denotes the probabilities of all combinations of the values of $Weather$ and $Cavity$. This is a 4×2 table of probabilities called the **joint probability distribution** of $Weather$ and $Cavity$. We can also mix variables with and without values; $\mathbf{P}(sunny, Cavity)$ would be a two-element vector giving the probabilities of a sunny day with a cavity and a sunny day with no cavity. The **P** notation makes certain expressions much more concise than they might otherwise be. For example, the product rules for all possible values of $Weather$ and $Cavity$ can be written as a single equation:

JOINT PROBABILITY DISTRIBUTION

$$\mathbf{P}(Weather, Cavity) = \mathbf{P}(Weather \mid Cavity)\mathbf{P}(Cavity) \ ,$$

instead of as these $4 \times 2 = 8$ equations (using abbreviations W and C):

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

As a degenerate case, $\mathbf{P}(\mathit{sunny}, \mathit{cavity})$ has no variables and thus is a one-element vector that is the probability of a sunny day with a cavity, which could also be written as $P(\mathit{sunny}, \mathit{cavity})$ or $P(\mathit{sunny} \wedge \mathit{cavity})$. We will sometimes use **P** notation to derive results about individual P values, and when we say "$\mathbf{P}(\mathit{sunny}) = 0.6$" it is really an abbreviation for "$\mathbf{P}(\mathit{sunny})$ is the one-element vector $\langle 0.6 \rangle$, which means that $P(\mathit{sunny}) = 0.6$."

Now we have defined a syntax for propositions and probability assertions and we have given part of the semantics: Equation (13.2) defines the probability of a proposition as the sum of the probabilities of worlds in which it holds. To complete the semantics, we need to say what the worlds are and how to determine whether a proposition holds in a world. We borrow this part directly from the semantics of propositional logic, as follows. *A possible world is defined to be an assignment of values to all of the random variables under consideration.* It is easy to see that this definition satisfies the basic requirement that possible worlds be mutually exclusive and exhaustive (Exercise 13.5). For example, if the random variables are Cavity, $\mathit{Toothache}$, and $\mathit{Weather}$, then there are $2 \times 2 \times 4 = 16$ possible worlds. Furthermore, the truth of any given proposition, no matter how complex, can be determined easily in such worlds using the same recursive definition of truth as for formulas in propositional logic.

From the preceding definition of possible worlds, it follows that a probability model is completely determined by the joint distribution for all of the random variables—the so-called **full joint probability distribution**. For example, if the variables are Cavity, $\mathit{Toothache}$, and $\mathit{Weather}$, then the full joint distribution is given by $\mathbf{P}(\mathit{Cavity}, \mathit{Toothache}, \mathit{Weather})$. This joint distribution can be represented as a $2 \times 2 \times 4$ table with 16 entries. Because every proposition's probability is a sum over possible worlds, a full joint distribution suffices, in principle, for calculating the probability of any proposition.

FULL JOINT PROBABILITY DISTRIBUTION

13.2.3 Probability axioms and their reasonableness

The basic axioms of probability (Equations (13.1) and (13.2)) imply certain relationships among the degrees of belief that can be accorded to logically related propositions. For example, we can derive the familiar relationship between the probability of a proposition and the probability of its negation:

$$
\begin{aligned}
P(\neg a) &= \textstyle\sum_{\omega \in \neg a} P(\omega) && \text{by Equation (13.2)} \\
&= \textstyle\sum_{\omega \in \neg a} P(\omega) + \sum_{\omega \in a} P(\omega) - \sum_{\omega \in a} P(\omega) && \\
&= \textstyle\sum_{\omega \in \Omega} P(\omega) - \sum_{\omega \in a} P(\omega) && \text{grouping the first two terms} \\
&= 1 - P(a) && \text{by (13.1) and (13.2).}
\end{aligned}
$$

INCLUSION–EXCLUSION PRINCIPLE

We can also derive the well-known formula for the probability of a disjunction, sometimes called the **inclusion–exclusion principle**:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b) \ . \tag{13.4}$$

This rule is easily remembered by noting that the cases where a holds, together with the cases where b holds, certainly cover all the cases where $a \vee b$ holds; but summing the two sets of cases counts their intersection twice, so we need to subtract $P(a \wedge b)$. The proof is left as an exercise (Exercise 13.6).

KOLMOGOROV'S AXIOMS

Equations (13.1) and (13.4) are often called **Kolmogorov's axioms** in honor of the Russian mathematician Andrei Kolmogorov, who showed how to build up the rest of probability theory from this simple foundation and how to handle the difficulties caused by continuous variables.[2] While Equation (13.2) has a definitional flavor, Equation (13.4) reveals that the axioms really do constrain the degrees of belief an agent can have concerning logically related propositions. This is analogous to the fact that a logical agent cannot simultaneously believe A, B, and $\neg(A \wedge B)$, because there is no possible world in which all three are true. With probabilities, however, statements refer not to the world directly, but to the agent's own state of knowledge. Why, then, can an agent not hold the following set of beliefs (even though they violate Kolmogorov's axioms)?

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

This kind of question has been the subject of decades of intense debate between those who advocate the use of probabilities as the only legitimate form for degrees of belief and those who advocate alternative approaches.

One argument for the axioms of probability, first stated in 1931 by Bruno de Finetti (and translated into English in de Finetti (1993)), is as follows: If an agent has some degree of belief in a proposition a, then the agent should be able to state odds at which it is indifferent to a bet for or against a.[3] Think of it as a game between two agents: Agent 1 states, "my degree of belief in event a is 0.4." Agent 2 is then free to choose whether to wager for or against a at stakes that are consistent with the stated degree of belief. That is, Agent 2 could choose to accept Agent 1's bet that a will occur, offering \$6 against Agent 1's \$4. Or Agent 2 could accept Agent 1's bet that $\neg a$ will occur, offering \$4 against Agent 1's \$6. Then we observe the outcome of a, and whoever is right collects the money. If an agent's degrees of belief do not accurately reflect the world, then you would expect that it would tend to lose money over the long run to an opposing agent whose beliefs more accurately reflect the state of the world.

But de Finetti proved something much stronger: *If Agent 1 expresses a set of degrees of belief that violate the axioms of probability theory then there is a combination of bets by Agent 2 that guarantees that Agent 1 will lose money every time.* For example, suppose that Agent 1 has the set of degrees of belief from Equation (13.5). Figure 13.2 shows that if Agent

[2] The difficulties include the **Vitali set**, a well-defined subset of the interval $[0, 1]$ with no well-defined size.

[3] One might argue that the agent's preferences for different bank balances are such that the possibility of losing \$1 is not counterbalanced by an equal possibility of winning \$1. One possible response is to make the bet amounts small enough to avoid this problem. Savage's analysis (1954) circumvents the issue altogether.

2 chooses to bet \$4 on a, \$3 on b, and \$2 on $\neg(a \vee b)$, then Agent 1 always loses money, regardless of the outcomes for a and b. De Finetti's theorem implies that no rational agent can have beliefs that violate the axioms of probability.

Agent 1		Agent 2		Outcomes and payoffs to Agent 1			
Proposition	Belief	Bet	Stakes	a, b	$a, \neg b$	$\neg a, b$	$\neg a, \neg b$
a	0.4	a	4 to 6	–6	–6	4	4
b	0.3	b	3 to 7	–7	3	–7	3
$a \vee b$	0.8	$\neg(a \vee b)$	2 to 8	2	2	2	–8
				–11	–1	–1	–1

Figure 13.2 Because Agent 1 has inconsistent beliefs, Agent 2 is able to devise a set of bets that guarantees a loss for Agent 1, no matter what the outcome of a and b.

One common objection to de Finetti's theorem is that this betting game is rather contrived. For example, what if one refuses to bet? Does that end the argument? The answer is that the betting game is an abstract model for the decision-making situation in which every agent is *unavoidably* involved at every moment. Every action (including inaction) is a kind of bet, and every outcome can be seen as a payoff of the bet. Refusing to bet is like refusing to allow time to pass.

Other strong philosophical arguments have been put forward for the use of probabilities, most notably those of Cox (1946), Carnap (1950), and Jaynes (2003). They each construct a set of axioms for reasoning with degrees of beliefs: no contradictions, correspondence with ordinary logic (for example, if belief in A goes up, then belief in $\neg A$ must go down), and so on. The only controversial axiom is that degrees of belief must be numbers, or at least act like numbers in that they must be transitive (if belief in A is greater than belief in B, which is greater than belief in C, then belief in A must be greater than C) and comparable (the belief in A must be one of equal to, greater than, or less than belief in B). It can then be proved that probability is the only approach that satisfies these axioms.

The world being the way it is, however, practical demonstrations sometimes speak louder than proofs. The success of reasoning systems based on probability theory has been much more effective in making converts. We now look at how the axioms can be deployed to make inferences.

13.3 Inference Using Full Joint Distributions

PROBABILISTIC INFERENCE

In this section we describe a simple method for **probabilistic inference**—that is, the computation of posterior probabilities for query propositions given observed evidence. We use the full joint distribution as the "knowledge base" from which answers to all questions may be derived. Along the way we also introduce several useful techniques for manipulating equations involving probabilities.

WHERE DO PROBABILITIES COME FROM?

There has been endless debate over the source and status of probability numbers. The **frequentist** position is that the numbers can come only from *experiments*: if we test 100 people and find that 10 of them have a cavity, then we can say that the probability of a cavity is approximately 0.1. In this view, the assertion "the probability of a cavity is 0.1" means that 0.1 is the fraction that would be observed in the limit of infinitely many samples. From any finite sample, we can estimate the true fraction and also calculate how accurate our estimate is likely to be.

The **objectivist** view is that probabilities are real aspects of the universe—propensities of objects to behave in certain ways—rather than being just descriptions of an observer's degree of belief. For example, the fact that a fair coin comes up heads with probability 0.5 is a propensity of the coin itself. In this view, frequentist measurements are attempts to observe these propensities. Most physicists agree that quantum phenomena are objectively probabilistic, but uncertainty at the macroscopic scale—e.g., in coin tossing—usually arises from ignorance of initial conditions and does not seem consistent with the propensity view.

The **subjectivist** view describes probabilities as a way of characterizing an agent's beliefs, rather than as having any external physical significance. The subjective **Bayesian** view allows any self-consistent ascription of prior probabilities to propositions, but then insists on proper Bayesian updating as evidence arrives.

In the end, even a strict frequentist position involves subjective analysis because of the **reference class** problem: in trying to determine the outcome probability of a *particular* experiment, the frequentist has to place it in a reference class of "similar" experiments with known outcome frequencies. I. J. Good (1983, p. 27) wrote, "every event in life is unique, and every real-life probability that we estimate in practice is that of an event that has never occurred before." For example, given a particular patient, a frequentist who wants to estimate the probability of a cavity will consider a reference class of other patients who are similar in important ways—age, symptoms, diet—and see what proportion of them had a cavity. If the dentist considers everything that is known about the patient—weight to the nearest gram, hair color, mother's maiden name—then the reference class becomes empty. This has been a vexing problem in the philosophy of science.

The **principle of indifference** attributed to Laplace (1816) states that propositions that are syntactically "symmetric" with respect to the evidence should be accorded equal probability. Various refinements have been proposed, culminating in the attempt by Carnap and others to develop a rigorous **inductive logic**, capable of computing the correct probability for any proposition from any collection of observations. Currently, it is believed that no unique inductive logic exists; rather, any such logic rests on a subjective prior probability distribution whose effect is diminished as more observations are collected.

	$toothache$		$\neg toothache$	
	$catch$	$\neg catch$	$catch$	$\neg catch$
$cavity$	0.108	0.012	0.072	0.008
$\neg cavity$	0.016	0.064	0.144	0.576

Figure 13.3 A full joint distribution for the $Toothache$, $Cavity$, $Catch$ world.

We begin with a simple example: a domain consisting of just the three Boolean variables $Toothache$, $Cavity$, and $Catch$ (the dentist's nasty steel probe catches in my tooth). The full joint distribution is a $2 \times 2 \times 2$ table as shown in Figure 13.3.

Notice that the probabilities in the joint distribution sum to 1, as required by the axioms of probability. Notice also that Equation (13.2) gives us a direct way to calculate the probability of any proposition, simple or complex: simply identify those possible worlds in which the proposition is true and add up their probabilities. For example, there are six possible worlds in which $cavity \vee toothache$ holds:

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

One particularly common task is to extract the distribution over some subset of variables or a single variable. For example, adding the entries in the first row gives the unconditional or

MARGINAL PROBABILITY

marginal probability[4] of $cavity$:

$$P(cavity) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2 \ .$$

MARGINALIZATION

This process is called **marginalization**, or **summing out**—because we sum up the probabilities for each possible value of the other variables, thereby taking them out of the equation. We can write the following general marginalization rule for any sets of variables $\mathbf{Y}$ and $\mathbf{Z}$:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z} \in \mathbf{Z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) \ , \tag{13.6}$$

where $\sum_{\mathbf{z} \in \mathbf{Z}}$ means to sum over all the possible combinations of values of the set of variables $\mathbf{Z}$. We sometimes abbreviate this as $\sum_{\mathbf{z}}$, leaving $\mathbf{Z}$ implicit. We just used the rule as

$$\mathbf{P}(Cavity) = \sum_{\mathbf{z} \in \{Catch, Toothache\}} \mathbf{P}(Cavity, \mathbf{z}) \ . \tag{13.7}$$

A variant of this rule involves conditional probabilities instead of joint probabilities, using the product rule:

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) P(\mathbf{z}) \ . \tag{13.8}$$

CONDITIONING

This rule is called **conditioning**. Marginalization and conditioning turn out to be useful rules for all kinds of derivations involving probability expressions.

In most cases, we are interested in computing *conditional* probabilities of some variables, given evidence about others. Conditional probabilities can be found by first using

[4] So called because of a common practice among actuaries of writing the sums of observed frequencies in the margins of insurance tables.

Equation (13.3) to obtain an expression in terms of unconditional probabilities and then evaluating the expression from the full joint distribution. For example, we can compute the probability of a cavity, given evidence of a toothache, as follows:

$$\begin{aligned} P(cavity \mid toothache) &= \frac{P(cavity \wedge toothache)}{P(toothache)} \\ &= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6 \ . \end{aligned}$$

Just to check, we can also compute the probability that there is no cavity, given a toothache:

$$\begin{aligned} P(\neg cavity \mid toothache) &= \frac{P(\neg cavity \wedge toothache)}{P(toothache)} \\ &= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4 \ . \end{aligned}$$

The two values sum to 1.0, as they should. Notice that in these two calculations the term $1/P(toothache)$ remains constant, no matter which value of $Cavity$ we calculate. In fact, it can be viewed as a **normalization** constant for the distribution $\mathbf{P}(Cavity \mid toothache)$, ensuring that it adds up to 1. Throughout the chapters dealing with probability, we use α to denote such constants. With this notation, we can write the two preceding equations in one:

NORMALIZATION

$$\begin{aligned} &\mathbf{P}(Cavity \mid toothache) = \alpha \, \mathbf{P}(Cavity, toothache) \\ &= \alpha \, [\mathbf{P}(Cavity, toothache, catch) + \mathbf{P}(Cavity, toothache, \neg catch)] \\ &= \alpha \, [\langle 0.108, 0.016 \rangle + \langle 0.012, 0.064 \rangle] = \alpha \, \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle \ . \end{aligned}$$

In other words, we can calculate $\mathbf{P}(Cavity \mid toothache)$ even if we don't know the value of $P(toothache)$! We temporarily forget about the factor $1/P(toothache)$ and add up the values for $cavity$ and $\neg cavity$, getting 0.12 and 0.08. Those are the correct relative proportions, but they don't sum to 1, so we normalize them by dividing each one by $0.12 + 0.08$, getting the true probabilities of 0.6 and 0.4. Normalization turns out to be a useful shortcut in many probability calculations, both to make the computation easier and to allow us to proceed when some probability assessment (such as $P(toothache)$) is not available.

From the example, we can extract a general inference procedure. We begin with the case in which the query involves a single variable, X ($Cavity$ in the example). Let $\mathbf{E}$ be the list of evidence variables (just $Toothache$ in the example), let $\mathbf{e}$ be the list of observed values for them, and let $\mathbf{Y}$ be the remaining unobserved variables (just $Catch$ in the example). The query is $\mathbf{P}(X \mid \mathbf{e})$ and can be evaluated as

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \ , \tag{13.9}$$

where the summation is over all possible $\mathbf{y}$s (i.e., all possible combinations of values of the unobserved variables $\mathbf{Y}$). Notice that together the variables X, $\mathbf{E}$, and $\mathbf{Y}$ constitute the complete set of variables for the domain, so $\mathbf{P}(X, \mathbf{e}, \mathbf{y})$ is simply a subset of probabilities from the full joint distribution.

Given the full joint distribution to work with, Equation (13.9) can answer probabilistic queries for discrete variables. It does not scale well, however: for a domain described by n Boolean variables, it requires an input table of size $O(2^n)$ and takes $O(2^n)$ time to process the

table. In a realistic problem we could easily have $n > 100$, making $O(2^n)$ impractical. The full joint distribution in tabular form is just not a practical tool for building reasoning systems. Instead, it should be viewed as the theoretical foundation on which more effective approaches may be built, just as truth tables formed a theoretical foundation for more practical algorithms like DPLL. The remainder of this chapter introduces some of the basic ideas required in preparation for the development of realistic systems in Chapter 14.

13.4 INDEPENDENCE

Let us expand the full joint distribution in Figure 13.3 by adding a fourth variable, $Weather$. The full joint distribution then becomes $\mathbf{P}(Toothache, Catch, Cavity, Weather)$, which has $2 \times 2 \times 2 \times 4 = 32$ entries. It contains four "editions" of the table shown in Figure 13.3, one for each kind of weather. What relationship do these editions have to each other and to the original three-variable table? For example, how are $P(toothache, catch, cavity, cloudy)$ and $P(toothache, catch, cavity)$ related? We can use the product rule:

$$\begin{aligned} &P(toothache, catch, cavity, cloudy) \\ &= P(cloudy \mid toothache, catch, cavity) P(toothache, catch, cavity) \ . \end{aligned}$$

Now, unless one is in the deity business, one should not imagine that one's dental problems influence the weather. And for indoor dentistry, at least, it seems safe to say that the weather does not influence the dental variables. Therefore, the following assertion seems reasonable:

$$P(cloudy \mid toothache, catch, cavity) = P(cloudy) \ . \tag{13.10}$$

From this, we can deduce

$$P(toothache, catch, cavity, cloudy) = P(cloudy) P(toothache, catch, cavity) \ .$$

A similar equation exists for *every entry* in $\mathbf{P}(Toothache, Catch, Cavity, Weather)$. In fact, we can write the general equation

$$\mathbf{P}(Toothache, Catch, Cavity, Weather) = \mathbf{P}(Toothache, Catch, Cavity)\mathbf{P}(Weather) \ .$$

Thus, the 32-element table for four variables can be constructed from one 8-element table and one 4-element table. This decomposition is illustrated schematically in Figure 13.4(a).

INDEPENDENCE

The property we used in Equation (13.10) is called **independence** (also **marginal independence** and **absolute independence**). In particular, the weather is independent of one's dental problems. Independence between propositions a and b can be written as

$$P(a \mid b) = P(a) \quad \text{or} \quad P(b \mid a) = P(b) \quad \text{or} \quad P(a \wedge b) = P(a) P(b) \ . \tag{13.11}$$

All these forms are equivalent (Exercise 13.12). Independence between variables X and Y can be written as follows (again, these are all equivalent):

$$\mathbf{P}(X \mid Y) = \mathbf{P}(X) \quad \text{or} \quad \mathbf{P}(Y \mid X) = \mathbf{P}(Y) \quad \text{or} \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y) \ .$$

Independence assertions are usually based on knowledge of the domain. As the toothache–weather example illustrates, they can dramatically reduce the amount of information necessary to specify the full joint distribution. If the complete set of variables can be divided

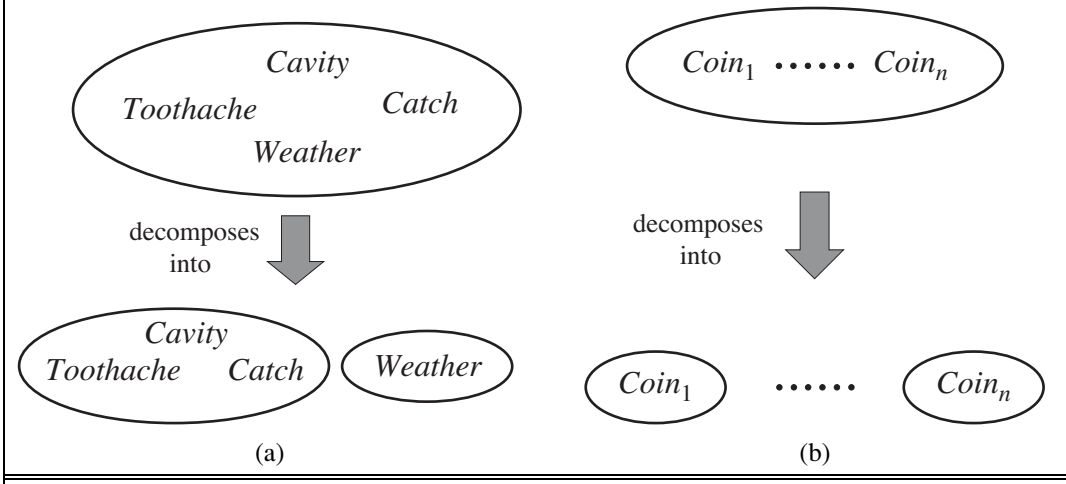


Figure 13.4 Two examples of factoring a large joint distribution into smaller distributions, using absolute independence. (a) Weather and dental problems are independent. (b) Coin flips are independent.

into independent subsets, then the full joint distribution can be *factored* into separate joint distributions on those subsets. For example, the full joint distribution on the outcome of n independent coin flips, $\mathbf{P}(C_1, \ldots, C_n)$, has 2^n entries, but it can be represented as the product of n single-variable distributions $\mathbf{P}(C_i)$. In a more practical vein, the independence of dentistry and meteorology is a good thing, because otherwise the practice of dentistry might require intimate knowledge of meteorology, and vice versa.

When they are available, then, independence assertions can help in reducing the size of the domain representation and the complexity of the inference problem. Unfortunately, clean separation of entire sets of variables by independence is quite rare. Whenever a connection, however indirect, exists between two variables, independence will fail to hold. Moreover, even independent subsets can be quite large—for example, dentistry might involve dozens of diseases and hundreds of symptoms, all of which are interrelated. To handle such problems, we need more subtle methods than the straightforward concept of independence.

13.5 Bayes' Rule and Its Use

On page 486, we defined the **product rule**. It can actually be written in two forms:

$$P(a \wedge b) = P(a \mid b)P(b) \qquad \text{and} \qquad P(a \wedge b) = P(b \mid a)P(a) \ .$$

Equating the two right-hand sides and dividing by $P(a)$, we get

$$P(b \mid a) = \frac{P(a \mid b)P(b)}{P(a)} \ . \tag{13.12}$$

BAYES' RULE

This equation is known as **Bayes' rule** (also Bayes' law or Bayes' theorem). This simple equation underlies most modern AI systems for probabilistic inference.

The more general case of Bayes' rule for multivalued variables can be written in the **P** notation as follows:

$$\mathbf{P}(Y \mid X) = \frac{\mathbf{P}(X \mid Y)\mathbf{P}(Y)}{\mathbf{P}(X)} ,$$

As before, this is to be taken as representing a set of equations, each dealing with specific values of the variables. We will also have occasion to use a more general version conditionalized on some background evidence **e**:

$$\mathbf{P}(Y \mid X, \mathbf{e}) = \frac{\mathbf{P}(X \mid Y, \mathbf{e})\mathbf{P}(Y \mid \mathbf{e})}{\mathbf{P}(X \mid \mathbf{e})} . \tag{13.13}$$

13.5.1 Applying Bayes' rule: The simple case

On the surface, Bayes' rule does not seem very useful. It allows us to compute the single term $P(b \mid a)$ in terms of three terms: $P(a \mid b)$, $P(b)$, and $P(a)$. That seems like two steps backwards, but Bayes' rule is useful in practice because there are many cases where we do have good probability estimates for these three numbers and need to compute the fourth. Often, we perceive as evidence the *effect* of some unknown *cause* and we would like to determine that cause. In that case, Bayes' rule becomes

$$P(\mathit{cause} \mid \mathit{effect}) = \frac{P(\mathit{effect} \mid \mathit{cause})P(\mathit{cause})}{P(\mathit{effect})} .$$

CAUSAL
DIAGNOSTIC

The conditional probability $P(\mathit{effect} \mid \mathit{cause})$ quantifies the relationship in the **causal** direction, whereas $P(\mathit{cause} \mid \mathit{effect})$ describes the **diagnostic** direction. In a task such as medical diagnosis, we often have conditional probabilities on causal relationships (that is, the doctor knows $P(\mathit{symptoms} \mid \mathit{disease})$) and want to derive a diagnosis, $P(\mathit{disease} \mid \mathit{symptoms})$. For example, a doctor knows that the disease meningitis causes the patient to have a stiff neck, say, 70% of the time. The doctor also knows some unconditional facts: the prior probability that a patient has meningitis is 1/50,000, and the prior probability that any patient has a stiff neck is 1%. Letting s be the proposition that the patient has a stiff neck and m be the proposition that the patient has meningitis, we have

$$\begin{aligned} P(s \mid m) &= 0.7 \\ P(m) &= 1/50000 \\ P(s) &= 0.01 \\ P(m \mid s) &= \frac{P(s \mid m)P(m)}{P(s)} = \frac{0.7 \times 1/50000}{0.01} = 0.0014 . \end{aligned} \tag{13.14}$$

That is, we expect less than 1 in 700 patients with a stiff neck to have meningitis. Notice that even though a stiff neck is quite strongly indicated by meningitis (with probability 0.7), the probability of meningitis in the patient remains small. This is because the prior probability of stiff necks is much higher than that of meningitis.

Section 13.3 illustrated a process by which one can avoid assessing the prior probability of the evidence (here, $P(s)$) by instead computing a posterior probability for each value of

the query variable (here, m and $\neg m$) and then normalizing the results. The same process can be applied when using Bayes' rule. We have

$$\mathbf{P}(M \mid s) = \alpha \langle P(s \mid m)P(m), P(s \mid \neg m)P(\neg m) \rangle \ .$$

Thus, to use this approach we need to estimate $P(s \mid \neg m)$ instead of $P(s)$. There is no free lunch—sometimes this is easier, sometimes it is harder. The general form of Bayes' rule with normalization is

$$\mathbf{P}(Y \mid X) = \alpha \mathbf{P}(X \mid Y)\mathbf{P}(Y) \ , \tag{13.15}$$

where α is the normalization constant needed to make the entries in $\mathbf{P}(Y \mid X)$ sum to 1.

One obvious question to ask about Bayes' rule is why one might have available the conditional probability in one direction, but not the other. In the meningitis domain, perhaps the doctor knows that a stiff neck implies meningitis in 1 out of 5000 cases; that is, the doctor has quantitative information in the **diagnostic** direction from symptoms to causes. Such a doctor has no need to use Bayes' rule. Unfortunately, *diagnostic knowledge is often more fragile than causal knowledge.* If there is a sudden epidemic of meningitis, the unconditional probability of meningitis, $P(m)$, will go up. The doctor who derived the diagnostic probability $P(m \mid s)$ directly from statistical observation of patients before the epidemic will have no idea how to update the value, but the doctor who computes $P(m \mid s)$ from the other three values will see that $P(m \mid s)$ should go up proportionately with $P(m)$. Most important, the causal information $P(s \mid m)$ is *unaffected* by the epidemic, because it simply reflects the way meningitis works. The use of this kind of direct causal or model-based knowledge provides the crucial robustness needed to make probabilistic systems feasible in the real world.

13.5.2 Using Bayes' rule: Combining evidence

We have seen that Bayes' rule can be useful for answering probabilistic queries conditioned on one piece of evidence—for example, the stiff neck. In particular, we have argued that probabilistic information is often available in the form $P(\textit{effect} \mid \textit{cause})$. What happens when we have two or more pieces of evidence? For example, what can a dentist conclude if her nasty steel probe catches in the aching tooth of a patient? If we know the full joint distribution (Figure 13.3), we can read off the answer:

$$\mathbf{P}(\textit{Cavity} \mid \textit{toothache} \wedge \textit{catch}) = \alpha \langle 0.108, 0.016 \rangle \approx \langle 0.871, 0.129 \rangle \ .$$

We know, however, that such an approach does not scale up to larger numbers of variables. We can try using Bayes' rule to reformulate the problem:

$$\begin{aligned} &\mathbf{P}(\textit{Cavity} \mid \textit{toothache} \wedge \textit{catch}) \\ &\quad = \alpha \mathbf{P}(\textit{toothache} \wedge \textit{catch} \mid \textit{Cavity})\, \mathbf{P}(\textit{Cavity}) \ . \end{aligned} \tag{13.16}$$

For this reformulation to work, we need to know the conditional probabilities of the conjunction $\textit{toothache} \wedge \textit{catch}$ for each value of $\textit{Cavity}$. That might be feasible for just two evidence variables, but again it does not scale up. If there are n possible evidence variables (X rays, diet, oral hygiene, etc.), then there are 2^n possible combinations of observed values for which we would need to know conditional probabilities. We might as well go back to using the full joint distribution. This is what first led researchers away from probability theory toward

approximate methods for evidence combination that, while giving incorrect answers, require fewer numbers to give any answer at all.

Rather than taking this route, we need to find some additional assertions about the domain that will enable us to simplify the expressions. The notion of **independence** in Section 13.4 provides a clue, but needs refining. It would be nice if $Toothache$ and $Catch$ were independent, but they are not: if the probe catches in the tooth, then it is likely that the tooth has a cavity and that the cavity causes a toothache. These variables *are* independent, however, *given the presence or the absence of a cavity*. Each is directly caused by the cavity, but neither has a direct effect on the other: toothache depends on the state of the nerves in the tooth, whereas the probe's accuracy depends on the dentist's skill, to which the toothache is irrelevant.[5] Mathematically, this property is written as

$$\mathbf{P}(toothache \wedge catch \mid Cavity) = \mathbf{P}(toothache \mid Cavity)\mathbf{P}(catch \mid Cavity) \; . \quad (13.17)$$

CONDITIONAL INDEPENDENCE

This equation expresses the **conditional independence** of $toothache$ and $catch$ given $Cavity$. We can plug it into Equation (13.16) to obtain the probability of a cavity:

$$\begin{aligned}&\mathbf{P}(Cavity \mid toothache \wedge catch) \\ &\quad = \alpha\, \mathbf{P}(toothache \mid Cavity)\, \mathbf{P}(catch \mid Cavity)\, \mathbf{P}(Cavity) \; . \end{aligned} \quad (13.18)$$

Now the information requirements are the same as for inference, using each piece of evidence separately: the prior probability $\mathbf{P}(Cavity)$ for the query variable and the conditional probability of each effect, given its cause.

The general definition of **conditional independence** of two variables X and Y, given a third variable Z, is

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\mathbf{P}(Y \mid Z) \; .$$

In the dentist domain, for example, it seems reasonable to assert conditional independence of the variables $Toothache$ and $Catch$, given $Cavity$:

$$\mathbf{P}(Toothache, Catch \mid Cavity) = \mathbf{P}(Toothache \mid Cavity)\mathbf{P}(Catch \mid Cavity) \; . \quad (13.19)$$

Notice that this assertion is somewhat stronger than Equation (13.17), which asserts independence only for specific values of $Toothache$ and $Catch$. As with absolute independence in Equation (13.11), the equivalent forms

$$\mathbf{P}(X \mid Y, Z) = \mathbf{P}(X \mid Z) \quad \text{and} \quad \mathbf{P}(Y \mid X, Z) = \mathbf{P}(Y \mid Z)$$

can also be used (see Exercise 13.17). Section 13.4 showed that absolute independence assertions allow a decomposition of the full joint distribution into much smaller pieces. It turns out that the same is true for conditional independence assertions. For example, given the assertion in Equation (13.19), we can derive a decomposition as follows:

$$\begin{aligned}&\mathbf{P}(Toothache, Catch, Cavity) \\ &\quad = \mathbf{P}(Toothache, Catch \mid Cavity)\mathbf{P}(Cavity) \quad \text{(product rule)} \\ &\quad = \mathbf{P}(Toothache \mid Cavity)\mathbf{P}(Catch \mid Cavity)\mathbf{P}(Cavity) \quad \text{(using 13.19).}\end{aligned}$$

(The reader can easily check that this equation does in fact hold in Figure 13.3.) In this way, the original large table is decomposed into three smaller tables. The original table has seven

[5] We assume that the patient and dentist are distinct individuals.

independent numbers ($2^3 = 8$ entries in the table, but they must sum to 1, so 7 are independent). The smaller tables contain five independent numbers (for a conditional probability distributions such as $\mathbf{P}(T|C$ there are two rows of two numbers, and each row sums to 1, so that's two independent numbers; for a prior distribution like $\mathbf{P}(C)$ there is only one independent number). Going from seven to five might not seem like a major triumph, but the point is that, for n symptoms that are all conditionally independent given $Cavity$, the size of the representation grows as $O(n)$ instead of $O(2^n)$. That means that *conditional independence assertions can allow probabilistic systems to scale up; moreover, they are much more commonly available than absolute independence assertions.* Conceptually, $Cavity$ **separates** $Toothache$ and $Catch$ because it is a direct cause of both of them. The decomposition of large probabilistic domains into weakly connected subsets through conditional independence is one of the most important developments in the recent history of AI.

SEPARATION

The dentistry example illustrates a commonly occurring pattern in which a single cause directly influences a number of effects, all of which are conditionally independent, given the cause. The full joint distribution can be written as

$$\mathbf{P}(Cause, Effect_1, \ldots, Effect_n) = \mathbf{P}(Cause) \prod_i \mathbf{P}(Effect_i \mid Cause) \ .$$

NAIVE BAYES

Such a probability distribution is called a **naive Bayes** model—"naive" because it is often used (as a simplifying assumption) in cases where the "effect" variables are *not* actually conditionally independent given the cause variable. (The naive Bayes model is sometimes called a **Bayesian classifier**, a somewhat careless usage that has prompted true Bayesians to call it the **idiot Bayes** model.) In practice, naive Bayes systems can work surprisingly well, even when the conditional independence assumption is not true. Chapter 20 describes methods for learning naive Bayes distributions from observations.

13.6 THE WUMPUS WORLD REVISITED

We can combine of the ideas in this chapter to solve probabilistic reasoning problems in the wumpus world. (See Chapter 7 for a complete description of the wumpus world.) Uncertainty arises in the wumpus world because the agent's sensors give only partial information about the world. For example, Figure 13.5 shows a situation in which each of the three reachable squares—[1,3], [2,2], and [3,1]—might contain a pit. Pure logical inference can conclude nothing about which square is most likely to be safe, so a logical agent might have to choose randomly. We will see that a probabilistic agent can do much better than the logical agent.

Our aim is to calculate the probability that each of the three squares contains a pit. (For this example we ignore the wumpus and the gold.) The relevant properties of the wumpus world are that (1) a pit causes breezes in all neighboring squares, and (2) each square other than [1,1] contains a pit with probability 0.2. The first step is to identify the set of random variables we need:

- As in the propositional logic case, we want one Boolean variable P_{ij} for each square, which is true iff square $[i, j]$ actually contains a pit.

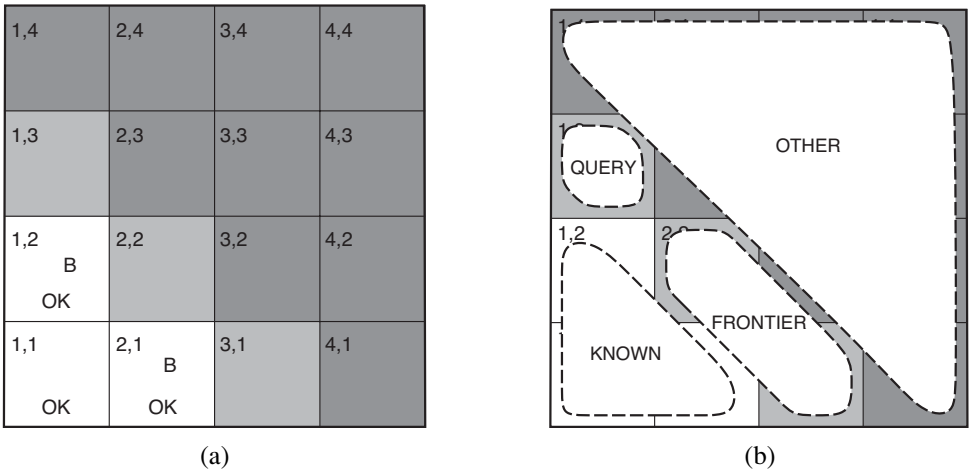


Figure 13.5 (a) After finding a breeze in both [1,2] and [2,1], the agent is stuck—there is no safe place to explore. (b) Division of the squares into $Known$, $Frontier$, and $Other$, for a query about [1,3].

- We also have Boolean variables B_{ij} that are true iff square $[i, j]$ is breezy; we include these variables only for the observed squares—in this case, [1,1], [1,2], and [2,1].

The next step is to specify the full joint distribution, $\mathbf{P}(P_{1,1}, \ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1})$. Applying the product rule, we have

$$\mathbf{P}(P_{1,1}, \ldots, P_{4,4}, B_{1,1}, B_{1,2}, B_{2,1}) = \\ \mathbf{P}(B_{1,1}, B_{1,2}, B_{2,1} \mid P_{1,1}, \ldots, P_{4,4})\mathbf{P}(P_{1,1}, \ldots, P_{4,4}) \, .$$

This decomposition makes it easy to see what the joint probability values should be. The first term is the conditional probability distribution of a breeze configuration, given a pit configuration; its values are 1 if the breezes are adjacent to the pits and 0 otherwise. The second term is the prior probability of a pit configuration. Each square contains a pit with probability 0.2, independently of the other squares; hence,

$$\mathbf{P}(P_{1,1}, \ldots, P_{4,4}) = \prod_{i,j=1,1}^{4,4} \mathbf{P}(P_{i,j}) \, . \tag{13.20}$$

For a particular configuration with exactly n pits, $P(P_{1,1}, \ldots, P_{4,4}) = 0.2^n \times 0.8^{16-n}$.

In the situation in Figure 13.5(a), the evidence consists of the observed breeze (or its absence) in each square that is visited, combined with the fact that each such square contains no pit. We abbreviate these facts as $b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$ and $known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$. We are interested in answering queries such as $\mathbf{P}(P_{1,3} \mid known, b)$: how likely is it that [1,3] contains a pit, given the observations so far?

To answer this query, we can follow the standard approach of Equation (13.9), namely, summing over entries from the full joint distribution. Let $Unknown$ be the set of $P_{i,j}$ vari-

ables for squares other than the $Known$ squares and the query square [1,3]. Then, by Equation (13.9), we have

$$\mathbf{P}(P_{1,3} \mid known, b) = \alpha \sum_{unknown} \mathbf{P}(P_{1,3}, unknown, known, b) \ .$$

The full joint probabilities have already been specified, so we are done—that is, unless we care about computation. There are 12 unknown squares; hence the summation contains $2^{12} = 4096$ terms. In general, the summation grows exponentially with the number of squares.

Surely, one might ask, aren't the other squares irrelevant? How could [4,4] affect whether [1,3] has a pit? Indeed, this intuition is correct. Let $Frontier$ be the pit variables (other than the query variable) that are adjacent to visited squares, in this case just [2,2] and [3,1]. Also, let $Other$ be the pit variables for the other unknown squares; in this case, there are 10 other squares, as shown in Figure 13.5(b). The key insight is that the observed breezes are *conditionally independent* of the other variables, given the known, frontier, and query variables. To use the insight, we manipulate the query formula into a form in which the breezes are conditioned on all the other variables, and then we apply conditional independence:

$$\begin{aligned}
&\mathbf{P}(P_{1,3} \mid known, b) \\
&= \alpha \sum_{unknown} \mathbf{P}(P_{1,3}, known, b, unknown) \qquad \text{(by Equation (13.9))} \\
&= \alpha \sum_{unknown} \mathbf{P}(b \mid P_{1,3}, known, unknown)\mathbf{P}(P_{1,3}, known, unknown) \\
&\qquad\qquad \text{(by the product rule)} \\
&= \alpha \sum_{frontier} \sum_{other} \mathbf{P}(b \mid known, P_{1,3}, frontier, other)\mathbf{P}(P_{1,3}, known, frontier, other) \\
&= \alpha \sum_{frontier} \sum_{other} \mathbf{P}(b \mid known, P_{1,3}, frontier)\mathbf{P}(P_{1,3}, known, frontier, other) \ ,
\end{aligned}$$

where the final step uses conditional independence: b is independent of $other$ given $known$, $P_{1,3}$, and $frontier$. Now, the first term in this expression does not depend on the $Other$ variables, so we can move the summation inward:

$$\begin{aligned}
&\mathbf{P}(P_{1,3} \mid known, b) \\
&= \alpha \sum_{frontier} \mathbf{P}(b \mid known, P_{1,3}, frontier) \sum_{other} \mathbf{P}(P_{1,3}, known, frontier, other) \ .
\end{aligned}$$

By independence, as in Equation (13.20), the prior term can be factored, and then the terms can be reordered:

$$\begin{aligned}
&\mathbf{P}(P_{1,3} \mid known, b) \\
&= \alpha \sum_{frontier} \mathbf{P}(b \mid known, P_{1,3}, frontier) \sum_{other} \mathbf{P}(P_{1,3})P(known)P(frontier)P(other) \\
&= \alpha \, P(known)\mathbf{P}(P_{1,3}) \sum_{frontier} \mathbf{P}(b \mid known, P_{1,3}, frontier)P(frontier) \sum_{other} P(other) \\
&= \alpha' \, \mathbf{P}(P_{1,3}) \sum_{frontier} \mathbf{P}(b \mid known, P_{1,3}, frontier)P(frontier) \ ,
\end{aligned}$$

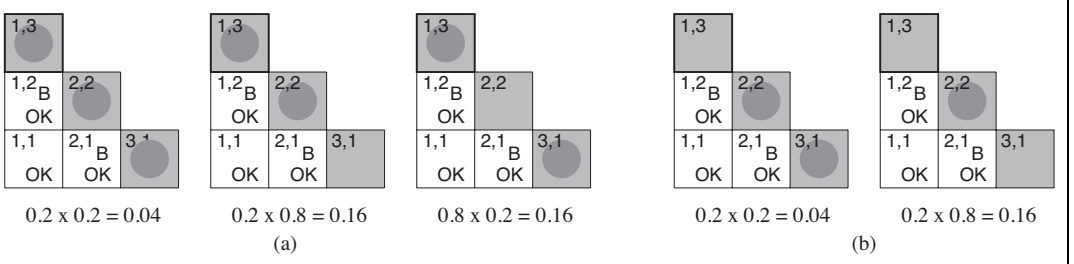


Figure 13.6 Consistent models for the frontier variables $P_{2,2}$ and $P_{3,1}$, showing $P(frontier)$ for each model: (a) three models with $P_{1,3} = true$ showing two or three pits, and (b) two models with $P_{1,3} = false$ showing one or two pits.

where the last step folds $P(known)$ into the normalizing constant and uses the fact that $\sum_{other} P(other)$ equals 1.

Now, there are just four terms in the summation over the frontier variables $P_{2,2}$ and $P_{3,1}$. The use of independence and conditional independence has completely eliminated the other squares from consideration.

Notice that the expression $\mathbf{P}(b \mid known, P_{1,3}, frontier)$ is 1 when the frontier is consistent with the breeze observations, and 0 otherwise. Thus, for each value of $P_{1,3}$, we sum over the *logical models* for the frontier variables that are consistent with the known facts. (Compare with the enumeration over models in Figure 7.5 on page 241.) The models and their associated prior probabilities—$P(frontier)$—are shown in Figure 13.6. We have

$$\mathbf{P}(P_{1,3} \mid known, b) = \alpha' \langle 0.2(0.04 + 0.16 + 0.16),\ 0.8(0.04 + 0.16) \rangle \approx \langle 0.31, 0.69 \rangle \ .$$

That is, [1,3] (and [3,1] by symmetry) contains a pit with roughly 31% probability. A similar calculation, which the reader might wish to perform, shows that [2,2] contains a pit with roughly 86% probability. The wumpus agent should definitely avoid [2,2]! Note that our logical agent from Chapter 7 did not know that [2,2] was worse than the other squares. Logic can tell us that it is unknown whether there is a pit in [2, 2], but we need probability to tell us how likely it is.

What this section has shown is that even seemingly complicated problems can be formulated precisely in probability theory and solved with simple algorithms. To get *efficient* solutions, independence and conditional independence relationships can be used to simplify the summations required. These relationships often correspond to our natural understanding of how the problem should be decomposed. In the next chapter, we develop formal representations for such relationships as well as algorithms that operate on those representations to perform probabilistic inference efficiently.

13.7 SUMMARY

This chapter has suggested probability theory as a suitable foundation for uncertain reasoning and provided a gentle introduction to its use.

- Uncertainty arises because of both laziness and ignorance. It is inescapable in complex, nondeterministic, or partially observable environments.
- Probabilities express the agent's inability to reach a definite decision regarding the truth of a sentence. Probabilities summarize the agent's beliefs relative to the evidence.
- Decision theory combines the agent's beliefs and desires, defining the best action as the one that maximizes expected utility.
- Basic probability statements include **prior probabilities** and **conditional probabilities** over simple and complex propositions.
- The axioms of probability constrain the possible assignments of probabilities to propositions. An agent that violates the axioms must behave irrationally in some cases.
- The **full joint probability distribution** specifies the probability of each complete assignment of values to random variables. It is usually too large to create or use in its explicit form, but when it is available it can be used to answer queries simply by adding up entries for the possible worlds corresponding to the query propositions.
- **Absolute independence** between subsets of random variables allows the full joint distribution to be factored into smaller joint distributions, greatly reducing its complexity. Absolute independence seldom occurs in practice.
- **Bayes' rule** allows unknown probabilities to be computed from known conditional probabilities, usually in the causal direction. Applying Bayes' rule with many pieces of evidence runs into the same scaling problems as does the full joint distribution.
- **Conditional independence** brought about by direct causal relationships in the domain might allow the full joint distribution to be factored into smaller, conditional distributions. The **naive Bayes** model assumes the conditional independence of all effect variables, given a single cause variable, and grows linearly with the number of effects.
- A wumpus-world agent can calculate probabilities for unobserved aspects of the world, thereby improving on the decisions of a purely logical agent. Conditional independence makes these calculations tractable.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Probability theory was invented as a way of analyzing games of chance. In about 850 A.D. the Indian mathematician Mahaviracarya described how to arrange a set of bets that can't lose (what we now call a Dutch book). In Europe, the first significant systematic analyses were produced by Girolamo Cardano around 1565, although publication was posthumous (1663). By that time, probability had been established as a mathematical discipline due to a series of

results established in a famous correspondence between Blaise Pascal and Pierre de Fermat in 1654. As with probability itself, the results were initially motivated by gambling problems (see Exercise 13.9). The first published textbook on probability was *De Ratiociniis in Ludo Aleae* (Huygens, 1657). The "laziness and ignorance" view of uncertainty was described by John Arbuthnot in the preface of his translation of Huygens (Arbuthnot, 1692): "It is impossible for a Die, with such determin'd force and direction, not to fall on such determin'd side, only I don't know the force and direction which makes it fall on such determin'd side, and therefore I call it Chance, which is nothing but the want of art..."

Laplace (1816) gave an exceptionally accurate and modern overview of probability; he was the first to use the example "take two urns, A and B, the first containing four white and two black balls, ..." The Rev. Thomas Bayes (1702–1761) introduced the rule for reasoning about conditional probabilities that was named after him (Bayes, 1763). Bayes only considered the case of uniform priors; it was Laplace who independently developed the general case. Kolmogorov (1950, first published in German in 1933) presented probability theory in a rigorously axiomatic framework for the first time. Rényi (1970) later gave an axiomatic presentation that took conditional probability, rather than absolute probability, as primitive.

Pascal used probability in ways that required both the objective interpretation, as a property of the world based on symmetry or relative frequency, and the subjective interpretation, based on degree of belief—the former in his analyses of probabilities in games of chance, the latter in the famous "Pascal's wager" argument about the possible existence of God. However, Pascal did not clearly realize the distinction between these two interpretations. The distinction was first drawn clearly by James Bernoulli (1654–1705).

Leibniz introduced the "classical" notion of probability as a proportion of enumerated, equally probable cases, which was also used by Bernoulli, although it was brought to prominence by Laplace (1749–1827). This notion is ambiguous between the frequency interpretation and the subjective interpretation. The cases can be thought to be equally probable either because of a natural, physical symmetry between them, or simply because we do not have any knowledge that would lead us to consider one more probable than another. The use of this latter, subjective consideration to justify assigning equal probabilities is known as the **principle of indifference**. The principle is often attributed to Laplace, but he never isolated the principle explicitly. George Boole and John Venn both referred to it as the **principle of insufficient reason**; the modern name is due to Keynes (1921).

PRINCIPLE OF INDIFFERENCE

PRINCIPLE OF INSUFFICIENT REASON

The debate between objectivists and subjectivists became sharper in the 20th century. Kolmogorov (1963), R. A. Fisher (1922), and Richard von Mises (1928) were advocates of the relative frequency interpretation. Karl Popper's (1959, first published in German in 1934) "propensity" interpretation traces relative frequencies to an underlying physical symmetry. Frank Ramsey (1931), Bruno de Finetti (1937), R. T. Cox (1946), Leonard Savage (1954), Richard Jeffrey (1983), and E. T. Jaynes (2003) interpreted probabilities as the degrees of belief of specific individuals. Their analyses of degree of belief were closely tied to utilities and to behavior—specifically, to the willingness to place bets. Rudolf Carnap, following Leibniz and Laplace, offered a different kind of subjective interpretation of probability—not as any actual individual's degree of belief, but as the degree of belief that an idealized individual *should* have in a particular proposition a, given a particular body of evidence **e**.

CONFIRMATION
INDUCTIVE LOGIC

Carnap attempted to go further than Leibniz or Laplace by making this notion of degree of **confirmation** mathematically precise, as a logical relation between a and **e**. The study of this relation was intended to constitute a mathematical discipline called **inductive logic**, analogous to ordinary deductive logic (Carnap, 1948, 1950). Carnap was not able to extend his inductive logic much beyond the propositional case, and Putnam (1963) showed by adversarial arguments that some fundamental difficulties would prevent a strict extension to languages capable of expressing arithmetic.

Cox's theorem (1946) shows that any system for uncertain reasoning that meets his set of assumptions is equivalent to probability theory. This gave renewed confidence to those who already favored probability, but others were not convinced, pointing to the assumptions (primarily that belief must be represented by a single number, and thus the belief in $\neg p$ must be a function of the belief in p). Halpern (1999) describes the assumptions and shows some gaps in Cox's original formulation. Horn (2003) shows how to patch up the difficulties. Jaynes (2003) has a similar argument that is easier to read.

The question of reference classes is closely tied to the attempt to find an inductive logic. The approach of choosing the "most specific" reference class of sufficient size was formally proposed by Reichenbach (1949). Various attempts have been made, notably by Henry Kyburg (1977, 1983), to formulate more sophisticated policies in order to avoid some obvious fallacies that arise with Reichenbach's rule, but such approaches remain somewhat *ad hoc*. More recent work by Bacchus, Grove, Halpern, and Koller (1992) extends Carnap's methods to first-order theories, thereby avoiding many of the difficulties associated with the straightforward reference-class method. Kyburg and Teng (2006) contrast probabilistic inference with nonmonotonic logic.

Bayesian probabilistic reasoning has been used in AI since the 1960s, especially in medical diagnosis. It was used not only to make a diagnosis from available evidence, but also to select further questions and tests by using the theory of information value (Section 16.6) when available evidence was inconclusive (Gorry, 1968; Gorry *et al.*, 1973). One system outperformed human experts in the diagnosis of acute abdominal illnesses (de Dombal *et al.*, 1974). Lucas *et al.* (2004) gives an overview. These early Bayesian systems suffered from a number of problems, however. Because they lacked any theoretical model of the conditions they were diagnosing, they were vulnerable to unrepresentative data occurring in situations for which only a small sample was available (de Dombal *et al.*, 1981). Even more fundamentally, because they lacked a concise formalism (such as the one to be described in Chapter 14) for representing and using conditional independence information, they depended on the acquisition, storage, and processing of enormous tables of probabilistic data. Because of these difficulties, probabilistic methods for coping with uncertainty fell out of favor in AI from the 1970s to the mid-1980s. Developments since the late 1980s are described in the next chapter.

The naive Bayes model for joint distributions has been studied extensively in the pattern recognition literature since the 1950s (Duda and Hart, 1973). It has also been used, often unwittingly, in information retrieval, beginning with the work of Maron (1961). The probabilistic foundations of this technique, described further in Exercise 13.22, were elucidated by Robertson and Sparck Jones (1976). Domingos and Pazzani (1997) provide an explanation

for the surprising success of naive Bayesian reasoning even in domains where the independence assumptions are clearly violated.

There are many good introductory textbooks on probability theory, including those by Bertsekas and Tsitsiklis (2008) and Grinstead and Snell (1997). DeGroot and Schervish (2001) offer a combined introduction to probability and statistics from a Bayesian standpoint. Richard Hamming's (1991) textbook gives a mathematically sophisticated introduction to probability theory from the standpoint of a propensity interpretation based on physical symmetry. Hacking (1975) and Hald (1990) cover the early history of the concept of probability. Bernstein (1996) gives an entertaining popular account of the story of risk.

EXERCISES

13.1 Show from first principles that $P(a \mid b \wedge a) = 1$.

13.2 Using the axioms of probability, prove that any probability distribution on a discrete random variable must sum to 1.

13.3 For each of the following statements, either prove it is true or give a counterexample.

a. If $P(a \mid b, c) = P(b \mid a, c)$, then $P(a \mid c) = P(b \mid c)$

b. If $P(a \mid b, c) = P(a)$, then $P(b \mid c) = P(b)$

c. If $P(a \mid b) = P(a)$, then $P(a \mid b, c) = P(a \mid c)$

13.4 Would it be rational for an agent to hold the three beliefs $P(A) = 0.4$, $P(B) = 0.3$, and $P(A \vee B) = 0.5$? If so, what range of probabilities would be rational for the agent to hold for $A \wedge B$? Make up a table like the one in Figure 13.2, and show how it supports your argument about rationality. Then draw another version of the table where $P(A \vee B) = 0.7$. Explain why it is rational to have this probability, even though the table shows one case that is a loss and three that just break even. (*Hint:* what is Agent 1 committed to about the probability of each of the four cases, especially the case that is a loss?)

13.5 This question deals with the properties of possible worlds, defined on page 488 as assignments to all random variables. We will work with propositions that correspond to exactly one possible world because they pin down the assignments of all the variables. In probability theory, such propositions are called **atomic events**. For example, with Boolean variables X_1, X_2, X_3, the proposition $x_1 \wedge \neg x_2 \wedge \neg x_3$ fixes the assignment of the variables; in the language of propositional logic, we would say it has exactly one model.

ATOMIC EVENT

a. Prove, for the case of n Boolean variables, that any two distinct atomic events are mutually exclusive; that is, their conjunction is equivalent to $false$.

b. Prove that the disjunction of all possible atomic events is logically equivalent to $true$.

c. Prove that any proposition is logically equivalent to the disjunction of the atomic events that entail its truth.

13.6 Prove Equation (13.4) from Equations (13.1) and (13.2).

13.7 Consider the set of all possible five-card poker hands dealt fairly from a standard deck of fifty-two cards.

a. How many atomic events are there in the joint probability distribution (i.e., how many five-card hands are there)?

b. What is the probability of each atomic event?

c. What is the probability of being dealt a royal straight flush? Four of a kind?

13.8 Given the full joint distribution shown in Figure 13.3, calculate the following:

a. $\mathbf{P}(toothache)$.

b. $\mathbf{P}(Cavity)$.

c. $\mathbf{P}(Toothache \mid cavity)$.

d. $\mathbf{P}(Cavity \mid toothache \vee catch)$.

13.9 In his letter of August 24, 1654, Pascal was trying to show how a pot of money should be allocated when a gambling game must end prematurely. Imagine a game where each turn consists of the roll of a die, player *E* gets a point when the die is even, and player *O* gets a point when the die is odd. The first player to get 7 points wins the pot. Suppose the game is interrupted with *E* leading 4–2. How should the money be fairly split in this case? What is the general formula? (Fermat and Pascal made several errors before solving the problem, but you should be able to get it right the first time.)

13.10 Deciding to put probability theory to good use, we encounter a slot machine with three independent wheels, each producing one of the four symbols BAR, BELL, LEMON, or CHERRY with equal probability. The slot machine has the following payout scheme for a bet of 1 coin (where "?" denotes that we don't care what comes up for that wheel):

BAR/BAR/BAR pays 20 coins
BELL/BELL/BELL pays 15 coins
LEMON/LEMON/LEMON pays 5 coins
CHERRY/CHERRY/CHERRY pays 3 coins
CHERRY/CHERRY/? pays 2 coins
CHERRY/?/? pays 1 coin

a. Compute the expected "payback" percentage of the machine. In other words, for each coin played, what is the expected coin return?

b. Compute the probability that playing the slot machine once will result in a win.

c. Estimate the mean and median number of plays you can expect to make until you go broke, if you start with 10 coins. You can run a simulation to estimate this, rather than trying to compute an exact answer.

13.11 We wish to transmit an n-bit message to a receiving agent. The bits in the message are independently corrupted (flipped) during transmission with ϵ probability each. With an extra parity bit sent along with the original information, a message can be corrected by the receiver

if at most one bit in the entire message (including the parity bit) has been corrupted. Suppose we want to ensure that the correct message is received with probability at least $1-\delta$. What is the maximum feasible value of n? Calculate this value for the case $\epsilon = 0.001$, $\delta = 0.01$.

13.12 Show that the three forms of independence in Equation (13.11) are equivalent.

13.13 Consider two medical tests, A and B, for a virus. Test A is 95% effective at recognizing the virus when it is present, but has a 10% false positive rate (indicating that the virus is present, when it is not). Test B is 90% effective at recognizing the virus, but has a 5% false positive rate. The two tests use independent methods of identifying the virus. The virus is carried by 1% of all people. Say that a person is tested for the virus using only one of the tests, and that test comes back positive for carrying the virus. Which test returning positive is more indicative of someone really carrying the virus? Justify your answer mathematically.

13.14 Suppose you are given a coin that lands *heads* with probability x and *tails* with probability $1-x$. Are the outcomes of successive flips of the coin independent of each other given that you know the value of x? Are the outcomes of successive flips of the coin independent of each other if you do *not* know the value of x? Justify your answer.

13.15 After your yearly checkup, the doctor has bad news and good news. The bad news is that you tested positive for a serious disease and that the test is 99% accurate (i.e., the probability of testing positive when you do have the disease is 0.99, as is the probability of testing negative when you don't have the disease). The good news is that this is a rare disease, striking only 1 in 10,000 people of your age. Why is it good news that the disease is rare? What are the chances that you actually have the disease?

13.16 It is quite often useful to consider the effect of some specific propositions in the context of some general background evidence that remains fixed, rather than in the complete absence of information. The following questions ask you to prove more general versions of the product rule and Bayes' rule, with respect to some background evidence **e**:

a. Prove the conditionalized version of the general product rule:

$$\mathbf{P}(X, Y \mid \mathbf{e}) = \mathbf{P}(X \mid Y, \mathbf{e})\mathbf{P}(Y \mid \mathbf{e}) \ .$$

b. Prove the conditionalized version of Bayes' rule in Equation (13.13).

13.17 Show that the statement of conditional independence

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\mathbf{P}(Y \mid Z)$$

is equivalent to each of the statements

$$\mathbf{P}(X \mid Y, Z) = \mathbf{P}(X \mid Z) \quad \text{and} \quad \mathbf{P}(B \mid X, Z) = \mathbf{P}(Y \mid Z) \ .$$

13.18 Suppose you are given a bag containing n unbiased coins. You are told that $n-1$ of these coins are normal, with heads on one side and tails on the other, whereas one coin is a fake, with heads on both sides.

a. Suppose you reach into the bag, pick out a coin at random, flip it, and get a head. What is the (conditional) probability that the coin you chose is the fake coin?

b. Suppose you continue flipping the coin for a total of k times after picking it and see k heads. Now what is the conditional probability that you picked the fake coin?

c. Suppose you wanted to decide whether the chosen coin was fake by flipping it k times. The decision procedure returns $fake$ if all k flips come up heads; otherwise it returns $normal$. What is the (unconditional) probability that this procedure makes an error?

13.19 In this exercise, you will complete the normalization calculation for the meningitis example. First, make up a suitable value for $P(s \mid \neg m)$, and use it to calculate unnormalized values for $P(m \mid s)$ and $P(\neg m \mid s)$ (i.e., ignoring the $P(s)$ term in the Bayes' rule expression, Equation (13.14)). Now normalize these values so that they add to 1.

13.20 Let X, Y, Z be Boolean random variables. Label the eight entries in the joint distribution $\mathbf{P}(X, Y, Z)$ as a through h. Express the statement that X and Y are conditionally independent given Z, as a set of equations relating a through h. How many *nonredundant* equations are there?

13.21 (Adapted from Pearl (1988).) Suppose you are a witness to a nighttime hit-and-run accident involving a taxi in Athens. All taxis in Athens are blue or green. You swear, under oath, that the taxi was blue. Extensive testing shows that, under the dim lighting conditions, discrimination between blue and green is 75% reliable.

a. Is it possible to calculate the most likely color for the taxi? (*Hint:* distinguish carefully between the proposition that the taxi *is* blue and the proposition that it *appears* blue.)

b. What if you know that 9 out of 10 Athenian taxis are green?

13.22 Text categorization is the task of assigning a given document to one of a fixed set of categories on the basis of the text it contains. Naive Bayes models are often used for this task. In these models, the query variable is the document category, and the "effect" variables are the presence or absence of each word in the language; the assumption is that words occur independently in documents, with frequencies determined by the document category.

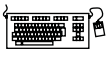
a. Explain precisely how such a model can be constructed, given as "training data" a set of documents that have been assigned to categories.

b. Explain precisely how to categorize a new document.

c. Is the conditional independence assumption reasonable? Discuss.

13.23 In our analysis of the wumpus world, we used the fact that each square contains a pit with probability 0.2, independently of the contents of the other squares. Suppose instead that exactly $N/5$ pits are scattered at random among the N squares other than [1,1]. Are the variables $P_{i,j}$ and $P_{k,l}$ still independent? What is the joint distribution $\mathbf{P}(P_{1,1}, \ldots, P_{4,4})$ now? Redo the calculation for the probabilities of pits in [1,3] and [2,2].

13.24 Redo the probability calculation for pits in [1,3] and [2,2], assuming that each square contains a pit with probability 0.01, independent of the other squares. What can you say about the relative performance of a logical versus a probabilistic agent in this case?



13.25 Implement a hybrid probabilistic agent for the wumpus world, based on the hybrid agent in Figure 7.20 and the probabilistic inference procedure outlined in this chapter.

14 PROBABILISTIC REASONING

In which we explain how to build network models to reason under uncertainty according to the laws of probability theory.

Chapter 13 introduced the basic elements of probability theory and noted the importance of independence and conditional independence relationships in simplifying probabilistic representations of the world. This chapter introduces a systematic way to represent such relationships explicitly in the form of **Bayesian networks**. We define the syntax and semantics of these networks and show how they can be used to capture uncertain knowledge in a natural and efficient way. We then show how probabilistic inference, although computationally intractable in the worst case, can be done efficiently in many practical situations. We also describe a variety of approximate inference algorithms that are often applicable when exact inference is infeasible. We explore ways in which probability theory can be applied to worlds with objects and relations—that is, to *first-order*, as opposed to *propositional*, representations. Finally, we survey alternative approaches to uncertain reasoning.

14.1 REPRESENTING KNOWLEDGE IN AN UNCERTAIN DOMAIN

In Chapter 13, we saw that the full joint probability distribution can answer any question about the domain, but can become intractably large as the number of variables grows. Furthermore, specifying probabilities for possible worlds one by one is unnatural and tedious.

We also saw that independence and conditional independence relationships among variables can greatly reduce the number of probabilities that need to be specified in order to define the full joint distribution. This section introduces a data structure called a **Bayesian network**[1] to represent the dependencies among variables. Bayesian networks can represent essentially *any* full joint probability distribution and in many cases can do so very concisely.

BAYESIAN NETWORK

[1] This is the most common name, but there are many synonyms, including **belief network**, **probabilistic network**, **causal network**, and **knowledge map**. In statistics, the term **graphical model** refers to a somewhat broader class that includes Bayesian networks. An extension of Bayesian networks called a **decision network** or **influence diagram** is covered in Chapter 16.

A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:

1. Each node corresponds to a random variable, which may be discrete or continuous.
2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a *parent* of Y. The graph has no directed cycles (and hence is a directed acyclic graph, or DAG.
3. Each node X_i has a conditional probability distribution $\mathbf{P}(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node.

The topology of the network—the set of nodes and links—specifies the conditional independence relationships that hold in the domain, in a way that will be made precise shortly. The *intuitive* meaning of an arrow is typically that X has a *direct influence* on Y, which suggests that causes should be parents of effects. It is usually easy for a domain expert to decide what direct influences exist in the domain—much easier, in fact, than actually specifying the probabilities themselves. Once the topology of the Bayesian network is laid out, we need only specify a conditional probability distribution for each variable, given its parents. We will see that the combination of the topology and the conditional distributions suffices to specify (implicitly) the full joint distribution for all the variables.

Recall the simple world described in Chapter 13, consisting of the variables $Toothache$, $Cavity$, $Catch$, and $Weather$. We argued that $Weather$ is independent of the other variables; furthermore, we argued that $Toothache$ and $Catch$ are conditionally independent, given $Cavity$. These relationships are represented by the Bayesian network structure shown in Figure 14.1. Formally, the conditional independence of $Toothache$ and $Catch$, given $Cavity$, is indicated by the *absence* of a link between $Toothache$ and $Catch$. Intuitively, the network represents the fact that $Cavity$ is a direct cause of $Toothache$ and $Catch$, whereas no direct causal relationship exists between $Toothache$ and $Catch$.

Now consider the following example, which is just a little more complex. You have a new burglar alarm installed at home. It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes. (This example is due to Judea Pearl, a resident of Los Angeles—hence the acute interest in earthquakes.) You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm. John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with

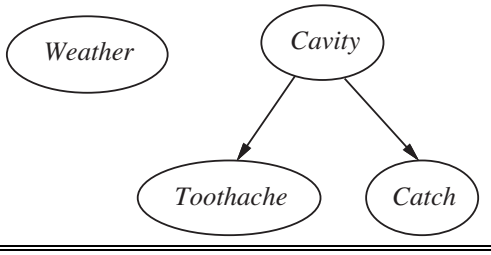


Figure 14.1 A simple Bayesian network in which $Weather$ is independent of the other three variables and $Toothache$ and $Catch$ are conditionally independent, given $Cavity$.

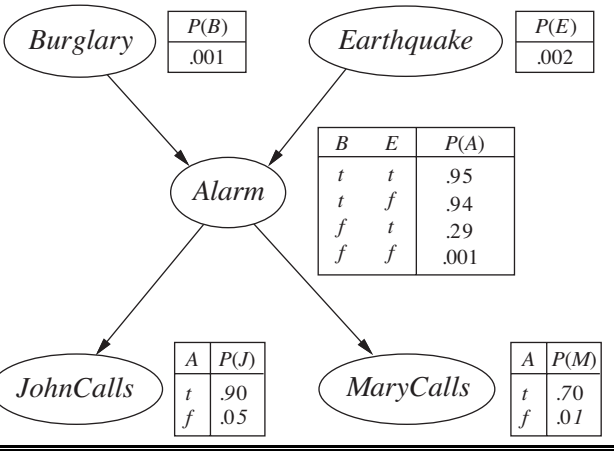


Figure 14.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for $Burglary$, $Earthquake$, $Alarm$, $JohnCalls$, and $MaryCalls$, respectively.

the alarm and calls then, too. Mary, on the other hand, likes rather loud music and often misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.

A Bayesian network for this domain appears in Figure 14.2. The network structure shows that burglary and earthquakes directly affect the probability of the alarm's going off, but whether John and Mary call depends only on the alarm. The network thus represents our assumptions that they do not perceive burglaries directly, they do not notice minor earthquakes, and they do not confer before calling.

CONDITIONAL PROBABILITY TABLE

CONDITIONING CASE

The conditional distributions in Figure 14.2 are shown as a **conditional probability table**, or CPT. (This form of table can be used for discrete variables; other representations, including those suitable for continuous variables, are described in Section 14.2.) Each row in a CPT contains the conditional probability of each node value for a **conditioning case**. A conditioning case is just a possible combination of values for the parent nodes—a miniature possible world, if you like. Each row must sum to 1, because the entries represent an exhaustive set of cases for the variable. For Boolean variables, once you know that the probability of a true value is p, the probability of false must be $1 - p$, so we often omit the second number, as in Figure 14.2. In general, a table for a Boolean variable with k Boolean parents contains 2^k independently specifiable probabilities. A node with no parents has only one row, representing the prior probabilities of each possible value of the variable.

Notice that the network does not have nodes corresponding to Mary's currently listening to loud music or to the telephone ringing and confusing John. These factors are summarized in the uncertainty associated with the links from $Alarm$ to $JohnCalls$ and $MaryCalls$. This shows both laziness and ignorance in operation: it would be a lot of work to find out why those factors would be more or less likely in any particular case, and we have no reasonable way to obtain the relevant information anyway. The probabilities actually summarize a *potentially*

infinite set of circumstances in which the alarm might fail to go off (high humidity, power failure, dead battery, cut wires, a dead mouse stuck inside the bell, etc.) or John or Mary might fail to call and report it (out to lunch, on vacation, temporarily deaf, passing helicopter, etc.). In this way, a small agent can cope with a very large world, at least approximately. The degree of approximation can be improved if we introduce additional relevant information.

14.2 THE SEMANTICS OF BAYESIAN NETWORKS

The previous section described what a network is, but not what it means. There are two ways in which one can understand the semantics of Bayesian networks. The first is to see the network as a representation of the joint probability distribution. The second is to view it as an encoding of a collection of conditional independence statements. The two views are equivalent, but the first turns out to be helpful in understanding how to *construct* networks, whereas the second is helpful in designing inference procedures.

14.2.1 Representing the full joint distribution

Viewed as a piece of "syntax," a Bayesian network is a directed acyclic graph with some numeric parameters attached to each node. One way to define what the network means—its semantics—is to define the way in which it represents a specific joint distribution over all the variables. To do this, we first need to retract (temporarily) what we said earlier about the parameters associated with each node. We said that those parameters correspond to conditional probabilities $\mathbf{P}(X_i \mid Parents(X_i))$; this is a true statement, but until we assign semantics to the network as a whole, we should think of them just as numbers $\theta(X_i \mid Parents(X_i))$.

A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. We use the notation $P(x_1, \ldots, x_n)$ as an abbreviation for this. The value of this entry is given by the formula

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} \theta(x_i \mid parents(X_i)) \,, \tag{14.1}$$

where $parents(X_i)$ denotes the values of $Parents(X_i)$ that appear in $x_1, \ldots, x_n$. Thus, each entry in the joint distribution is represented by the product of the appropriate elements of the conditional probability tables (CPTs) in the Bayesian network.

From this definition, it is easy to prove that the parameters $\theta(X_i \mid Parents(X_i))$ are exactly the conditional probabilities $\mathbf{P}(X_i \mid Parents(X_i))$ implied by the joint distribution (see Exercise 14.2). Hence, we can rewrite Equation (14.1) as

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i)) \,. \tag{14.2}$$

In other words, the tables we have been calling conditional probability tables really *are* conditional probability tables according to the semantics defined in Equation (14.1).

To illustrate this, we can calculate the probability that the alarm has sounded, but neither a burglary nor an earthquake has occurred, and both John and Mary call. We multiply entries

from the joint distribution (using single-letter names for the variables):

$$\begin{aligned} P(j, m, a, \neg b, \neg e) &= P(j \mid a)P(m \mid a)P(a \mid \neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.000628 \ . \end{aligned}$$

Section 13.3 explained that the full joint distribution can be used to answer any query about the domain. If a Bayesian network is a representation of the joint distribution, then it too can be used to answer any query, by summing all the relevant joint entries. Section 14.4 explains how to do this, but also describes methods that are much more efficient.

A method for constructing Bayesian networks

Equation (14.2) defines what a given Bayesian network means. The next step is to explain how to *construct* a Bayesian network in such a way that the resulting joint distribution is a good representation of a given domain. We will now show that Equation (14.2) implies certain conditional independence relationships that can be used to guide the knowledge engineer in constructing the topology of the network. First, we rewrite the entries in the joint distribution in terms of conditional probability, using the product rule (see page 486):

$$P(x_1, \ldots, x_n) = P(x_n \mid x_{n-1}, \ldots, x_1)P(x_{n-1}, \ldots, x_1) \ .$$

Then we repeat the process, reducing each conjunctive probability to a conditional probability and a smaller conjunction. We end up with one big product:

$$\begin{aligned} P(x_1, \ldots, x_n) &= P(x_n \mid x_{n-1}, \ldots, x_1)P(x_{n-1} \mid x_{n-2}, \ldots, x_1) \ \cdots \ P(x_2 \mid x_1)P(x_1) \\ &= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) \ . \end{aligned}$$

CHAIN RULE

This identity is called the **chain rule**. It holds for any set of random variables. Comparing it with Equation (14.2), we see that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i in the network,

$$\mathbf{P}(X_i \mid X_{i-1}, \ldots, X_1) = \mathbf{P}(X_i \mid Parents(X_i)) \ , \tag{14.3}$$

provided that $Parents(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$. This last condition is satisfied by numbering the nodes in a way that is consistent with the partial order implicit in the graph structure.

What Equation (14.3) says is that the Bayesian network is a correct representation of the domain only if each node is conditionally independent of its other predecessors in the node ordering, given its parents. We can satisfy this condition with this methodology:

1. *Nodes:* First determine the set of variables that are required to model the domain. Now order them, $\{X_1, \ldots, X_n\}$. Any order will work, but the resulting network will be more compact if the variables are ordered such that causes precede effects.
2. *Links:* For i = 1 to n do:
   - Choose, from $X_1, \ldots, X_{i-1}$, a minimal set of parents for X_i, such that Equation (14.3) is satisfied.
   - For each parent insert a link from the parent to X_i.
   - CPTs: Write down the conditional probability table, $\mathbf{P}(X_i|Parents(X_i))$.

Intuitively, the parents of node X_i should contain all those nodes in $X_1, \ldots, X_{i-1}$ that *directly influence* X_i. For example, suppose we have completed the network in Figure 14.2 except for the choice of parents for $MaryCalls$. $MaryCalls$ is certainly influenced by whether there is a $Burglary$ or an $Earthquake$, but not *directly* influenced. Intuitively, our knowledge of the domain tells us that these events influence Mary's calling behavior only through their effect on the alarm. Also, given the state of the alarm, whether John calls has no influence on Mary's calling. Formally speaking, we believe that the following conditional independence statement holds:

$$\mathbf{P}(MaryCalls \mid JohnCalls, Alarm, Earthquake, Burglary) = \mathbf{P}(MaryCalls \mid Alarm) \ .$$

Thus, $Alarm$ will be the only parent node for $MaryCalls$.

Because each node is connected only to earlier nodes, this construction method guarantees that the network is acyclic. Another important property of Bayesian networks is that they contain no redundant probability values. If there is no redundancy, then there is no chance for inconsistency: *it is impossible for the knowledge engineer or domain expert to create a Bayesian network that violates the axioms of probability.*

Compactness and node ordering

LOCALLY STRUCTURED
SPARSE

As well as being a complete and nonredundant representation of the domain, a Bayesian network can often be far more *compact* than the full joint distribution. This property is what makes it feasible to handle domains with many variables. The compactness of Bayesian networks is an example of a general property of **locally structured** (also called **sparse**) systems. In a locally structured system, each subcomponent interacts directly with only a bounded number of other components, regardless of the total number of components. Local structure is usually associated with linear rather than exponential growth in complexity. In the case of Bayesian networks, it is reasonable to suppose that in most domains each random variable is directly influenced by at most k others, for some constant k. If we assume n Boolean variables for simplicity, then the amount of information needed to specify each conditional probability table will be at most 2^k numbers, and the complete network can be specified by $n2^k$ numbers. In contrast, the joint distribution contains 2^n numbers. To make this concrete, suppose we have $n = 30$ nodes, each with five parents ($k = 5$). Then the Bayesian network requires 960 numbers, but the full joint distribution requires over a billion.

There are domains in which each variable can be influenced directly by all the others, so that the network is fully connected. Then specifying the conditional probability tables requires the same amount of information as specifying the joint distribution. In some domains, there will be slight dependencies that should strictly be included by adding a new link. But if these dependencies are tenuous, then it may not be worth the additional complexity in the network for the small gain in accuracy. For example, one might object to our burglary network on the grounds that if there is an earthquake, then John and Mary would not call even if they heard the alarm, because they assume that the earthquake is the cause. Whether to add the link from $Earthquake$ to $JohnCalls$ and $MaryCalls$ (and thus enlarge the tables) depends on comparing the importance of getting more accurate probabilities with the cost of specifying the extra information.

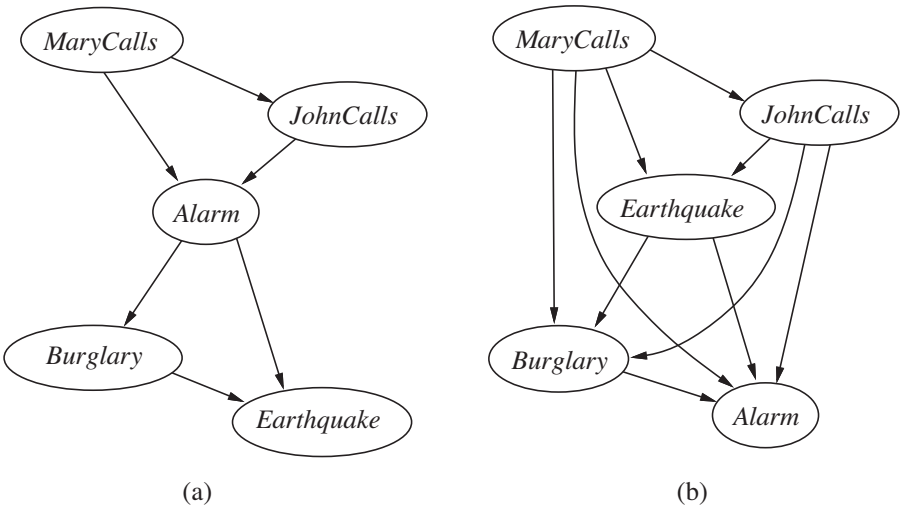


Figure 14.3 Network structure depends on order of introduction. In each network, we have introduced nodes in top-to-bottom order.

Even in a locally structured domain, we will get a compact Bayesian network only if we choose the node ordering well. What happens if we happen to choose the wrong order? Consider the burglary example again. Suppose we decide to add the nodes in the order $MaryCalls$, $JohnCalls$, $Alarm$, $Burglary$, $Earthquake$. We then get the somewhat more complicated network shown in Figure 14.3(a). The process goes as follows:

- Adding $MaryCalls$: No parents.
- Adding $JohnCalls$: If Mary calls, that probably means the alarm has gone off, which of course would make it more likely that John calls. Therefore, $JohnCalls$ needs $MaryCalls$ as a parent.
- Adding $Alarm$: Clearly, if both call, it is more likely that the alarm has gone off than if just one or neither calls, so we need both $MaryCalls$ and $JohnCalls$ as parents.
- Adding $Burglary$: If we know the alarm state, then the call from John or Mary might give us information about our phone ringing or Mary's music, but not about burglary:

  $$\mathbf{P}(Burglary \mid Alarm, JohnCalls, MaryCalls) = \mathbf{P}(Burglary \mid Alarm) \ .$$

  Hence we need just $Alarm$ as parent.
- Adding $Earthquake$: If the alarm is on, it is more likely that there has been an earthquake. (The alarm is an earthquake detector of sorts.) But if we know that there has been a burglary, then that explains the alarm, and the probability of an earthquake would be only slightly above normal. Hence, we need both $Alarm$ and $Burglary$ as parents.

The resulting network has two more links than the original network in Figure 14.2 and requires three more probabilities to be specified. What's worse, some of the links represent tenuous relationships that require difficult and unnatural probability judgments, such as as-

sessing the probability of $Earthquake$, given $Burglary$ and $Alarm$. This phenomenon is quite general and is related to the distinction between **causal** and **diagnostic** models introduced in Section 13.5.1 (see also Exercise 8.13). If we try to build a diagnostic model with links from symptoms to causes (as from $MaryCalls$ to $Alarm$ or $Alarm$ to $Burglary$), we end up having to specify additional dependencies between otherwise independent causes (and often between separately occurring symptoms as well). *If we stick to a causal model, we end up having to specify fewer numbers, and the numbers will often be easier to come up with.* In the domain of medicine, for example, it has been shown by Tversky and Kahneman (1982) that expert physicians prefer to give probability judgments for causal rules rather than for diagnostic ones.

Figure 14.3(b) shows a very bad node ordering: $MaryCalls$, $JohnCalls$, $Earthquake$, $Burglary$, $Alarm$. This network requires 31 distinct probabilities to be specified—exactly the same number as the full joint distribution. It is important to realize, however, that any of the three networks can represent *exactly the same joint distribution*. The last two versions simply fail to represent all the conditional independence relationships and hence end up specifying a lot of unnecessary numbers instead.

14.2.2 Conditional independence relations in Bayesian networks

We have provided a "numerical" semantics for Bayesian networks in terms of the representation of the full joint distribution, as in Equation (14.2). Using this semantics to derive a method for constructing Bayesian networks, we were led to the consequence that a node is conditionally independent of its other predecessors, given its parents. It turns out that we can also go in the other direction. We can start from a "topological" semantics that specifies the conditional independence relationships encoded by the graph structure, and from this we can derive the "numerical" semantics. The topological semantics[2] specifies that each variable is conditionally independent of its non-**descendants**, given its parents. For example, in Figure 14.2, $JohnCalls$ is independent of $Burglary$, $Earthquake$, and $MaryCalls$ given the value of $Alarm$. The definition is illustrated in Figure 14.4(a). From these conditional independence assertions and the interpretation of the network parameters $\theta(X_i \mid Parents(X_i))$ as specifications of conditional probabilities $\mathbf{P}(X_i \mid Parents(X_i))$, the full joint distribution given in Equation (14.2) can be reconstructed. In this sense, the "numerical" semantics and the "topological" semantics are equivalent.

DESCENDANT

Another important independence property is implied by the topological semantics: a node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents—that is, given its **Markov blanket**. (Exercise 14.7 asks you to prove this.) For example, $Burglary$ is independent of $JohnCalls$ and $MaryCalls$, given $Alarm$ and $Earthquake$. This property is illustrated in Figure 14.4(b).

MARKOV BLANKET

[2] There is also a general topological criterion called **d-separation** for deciding whether a set of nodes **X** is conditionally independent of another set **Y**, given a third set **Z**. The criterion is rather complicated and is not needed for deriving the algorithms in this chapter, so we omit it. Details may be found in Pearl (1988) or Darwiche (2009). Shachter (1998) gives a more intuitive method of ascertaining d-separation.

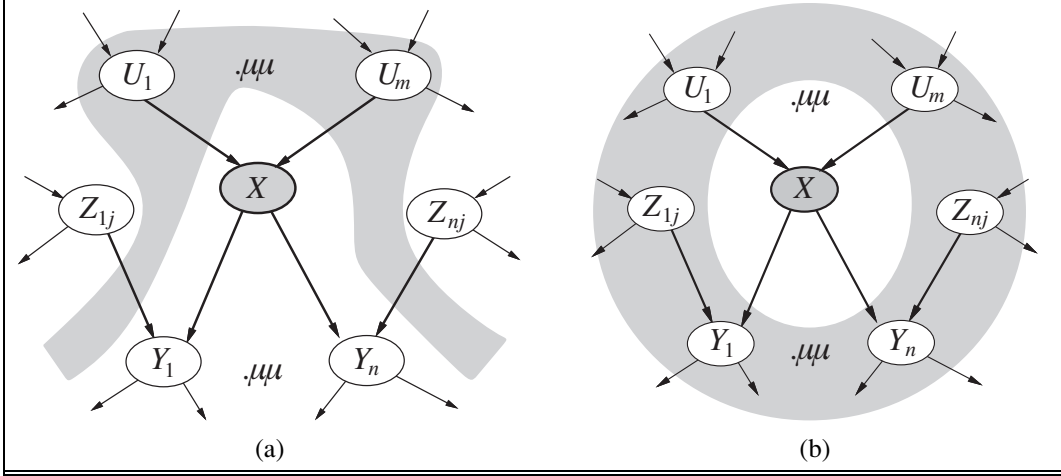


Figure 14.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the gray area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the gray area).

14.3 Efficient Representation of Conditional Distributions

Even if the maximum number of parents k is smallish, filling in the CPT for a node requires up to $O(2^k)$ numbers and perhaps a great deal of experience with all the possible conditioning cases. In fact, this is a worst-case scenario in which the relationship between the parents and the child is completely arbitrary. Usually, such relationships are describable by a **canonical distribution** that fits some standard pattern. In such cases, the complete table can be specified by naming the pattern and perhaps supplying a few parameters—much easier than supplying an exponential number of parameters.

CANONICAL DISTRIBUTION

DETERMINISTIC NODES

The simplest example is provided by **deterministic nodes**. A deterministic node has its value specified exactly by the values of its parents, with no uncertainty. The relationship can be a logical one: for example, the relationship between the parent nodes $Canadian$, US, $Mexican$ and the child node $NorthAmerican$ is simply that the child is the disjunction of the parents. The relationship can also be numerical: for example, if the parent nodes are the prices of a particular model of car at several dealers and the child node is the price that a bargain hunter ends up paying, then the child node is the minimum of the parent values; or if the parent nodes are a lake's inflows (rivers, runoff, precipitation) and outflows (rivers, evaporation, seepage) and the child is the change in the water level of the lake, then the value of the child is the sum of the inflow parents minus the sum of the outflow parents.

NOISY-OR

Uncertain relationships can often be characterized by so-called **noisy** logical relationships. The standard example is the **noisy-OR** relation, which is a generalization of the logical OR. In propositional logic, we might say that $Fever$ is true if and only if $Cold$, Flu, or $Malaria$ is true. The noisy-OR model allows for uncertainty about the ability of each parent to cause the child to be true—the causal relationship between parent and child may be

inhibited, and so a patient could have a cold, but not exhibit a fever. The model makes two assumptions. First, it assumes that all the possible causes are listed. (If some are missing, we can always add a so-called **leak node** that covers "miscellaneous causes.") Second, it assumes that inhibition of each parent is independent of inhibition of any other parents: for example, whatever inhibits $Malaria$ from causing a fever is independent of whatever inhibits Flu from causing a fever. Given these assumptions, $Fever$ is *false* if and only if all its *true* parents are inhibited, and the probability of this is the product of the inhibition probabilities q for each parent. Let us suppose these individual inhibition probabilities are as follows:

LEAK NODE

$$q_{\text{cold}} = P(\neg fever \mid cold, \neg flu, \neg malaria) = 0.6 ,$$
$$q_{\text{flu}} = P(\neg fever \mid \neg cold, flu, \neg malaria) = 0.2 ,$$
$$q_{\text{malaria}} = P(\neg fever \mid \neg cold, \neg flu, malaria) = 0.1 .$$

Then, from this information and the noisy-OR assumptions, the entire CPT can be built. The general rule is that

$$P(x_i \mid parents(X_i)) = 1 - \prod_{\{j : X_j = true\}} q_j ,$$

where the product is taken over the parents that are set to true for that row of the CPT. The following table illustrates this calculation:

$Cold$	Flu	$Malaria$	$P(Fever)$	$P(\neg Fever)$
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	$0.012 = 0.6 \times 0.2 \times 0.1$

In general, noisy logical relationships in which a variable depends on k parents can be described using $O(k)$ parameters instead of $O(2^k)$ for the full conditional probability table. This makes assessment and learning much easier. For example, the CPCS network (Pradhan *et al.*, 1994) uses noisy-OR and noisy-MAX distributions to model relationships among diseases and symptoms in internal medicine. With 448 nodes and 906 links, it requires only 8,254 values instead of 133,931,430 for a network with full CPTs.

Bayesian nets with continuous variables

Many real-world problems involve continuous quantities, such as height, mass, temperature, and money; in fact, much of statistics deals with random variables whose domains are continuous. By definition, continuous variables have an infinite number of possible values, so it is impossible to specify conditional probabilities explicitly for each value. One possible way to handle continuous variables is to avoid them by using **discretization**—that is, dividing up the

DISCRETIZATION

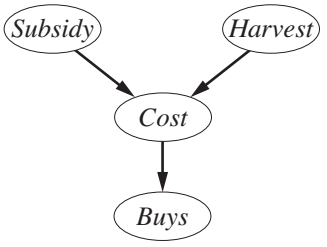


Figure 14.5 A simple network with discrete variables ($Subsidy$ and $Buys$) and continuous variables ($Harvest$ and $Cost$).

possible values into a fixed set of intervals. For example, temperatures could be divided into ($<0^{\circ}$C), (0°C-100°C), and ($>100^{\circ}$C). Discretization is sometimes an adequate solution, but often results in a considerable loss of accuracy and very large CPTs. The most common solution is to define standard families of probability density functions (see Appendix A) that are specified by a finite number of **parameters**. For example, a Gaussian (or normal) distribution $N(\mu, \sigma^2)(x)$ has the mean μ and the variance σ^2 as parameters. Yet another solution—sometimes called a **nonparametric** representation—is to define the conditional distribution implicitly with a collection of instances, each containing specific values of the parent and child variables. We explore this approach further in Chapter 18.

PARAMETER

NONPARAMETRIC

A network with both discrete and continuous variables is called a **hybrid Bayesian network**. To specify a hybrid network, we have to specify two new kinds of distributions: the conditional distribution for a continuous variable given discrete or continuous parents; and the conditional distribution for a discrete variable given continuous parents. Consider the simple example in Figure 14.5, in which a customer buys some fruit depending on its cost, which depends in turn on the size of the harvest and whether the government's subsidy scheme is operating. The variable $Cost$ is continuous and has continuous and discrete parents; the variable $Buys$ is discrete and has a continuous parent.

HYBRID BAYESIAN NETWORK

For the $Cost$ variable, we need to specify $\mathbf{P}(Cost \mid Harvest, Subsidy)$. The discrete parent is handled by enumeration—that is, by specifying both $P(Cost \mid Harvest, subsidy)$ and $P(Cost \mid Harvest, \neg subsidy)$. To handle $Harvest$, we specify how the distribution over the cost c depends on the continuous value h of $Harvest$. In other words, we specify the *parameters* of the cost distribution as a function of h. The most common choice is the **linear Gaussian** distribution, in which the child has a Gaussian distribution whose mean μ varies linearly with the value of the parent and whose standard deviation σ is fixed. We need two distributions, one for $subsidy$ and one for $\neg subsidy$, with different parameters:

LINEAR GAUSSIAN

$$P(c \mid h, subsidy) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_t h + b_t)}{\sigma_t}\right)^2}$$

$$P(c \mid h, \neg subsidy) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_f h + b_f)}{\sigma_f}\right)^2} .$$

For this example, then, the conditional distribution for $Cost$ is specified by naming the linear Gaussian distribution and providing the parameters a_t, b_t, σ_t, a_f, b_f, and σ_f. Figures 14.6(a)

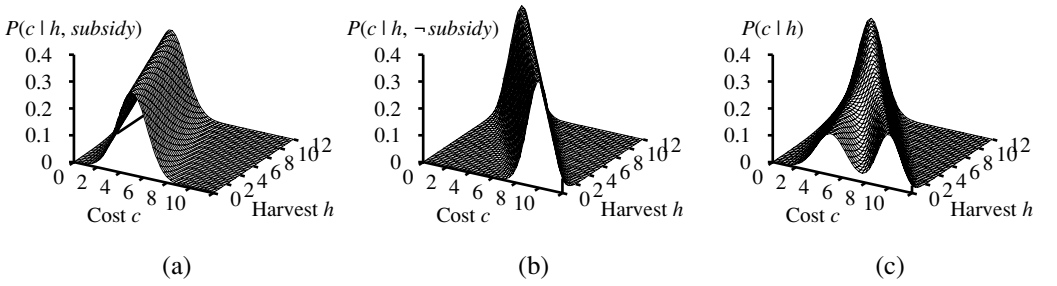


Figure 14.6 The graphs in (a) and (b) show the probability distribution over $Cost$ as a function of $Harvest$ size, with $Subsidy$ true and false, respectively. Graph (c) shows the distribution $P(Cost \mid Harvest)$, obtained by summing over the two subsidy cases.

and (b) show these two relationships. Notice that in each case the slope is negative, because cost decreases as supply increases. (Of course, the assumption of linearity implies that the cost becomes negative at some point; the linear model is reasonable only if the harvest size is limited to a narrow range.) Figure 14.6(c) shows the distribution $P(c \mid h)$, averaging over the two possible values of $Subsidy$ and assuming that each has prior probability 0.5. This shows that even with very simple models, quite interesting distributions can be represented.

The linear Gaussian conditional distribution has some special properties. A network containing only continuous variables with linear Gaussian distributions has a joint distribution that is a multivariate Gaussian distribution (see Appendix A) over all the variables (Exercise 14.9). Furthermore, the posterior distribution given any evidence also has this property.[3] When discrete variables are added as parents (not as children) of continuous variables, the network defines a **conditional Gaussian**, or CG, distribution: given any assignment to the discrete variables, the distribution over the continuous variables is a multivariate Gaussian.

CONDITIONAL GAUSSIAN

Now we turn to the distributions for discrete variables with continuous parents. Consider, for example, the $Buys$ node in Figure 14.5. It seems reasonable to assume that the customer will buy if the cost is low and will not buy if it is high and that the probability of buying varies smoothly in some intermediate region. In other words, the conditional distribution is like a "soft" threshold function. One way to make soft thresholds is to use the *integral* of the standard normal distribution:

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx \ .$$

Then the probability of $Buys$ given $Cost$ might be

$$P(buys \mid Cost = c) = \Phi((-c + \mu)/\sigma) \ ,$$

which means that the cost threshold occurs around μ, the width of the threshold region is proportional to σ, and the probability of buying decreases as cost increases. This **probit distri-**

[3] It follows that inference in linear Gaussian networks takes only $O(n^3)$ time in the worst case, regardless of the network topology. In Section 14.4, we see that inference for networks of discrete variables is NP-hard.

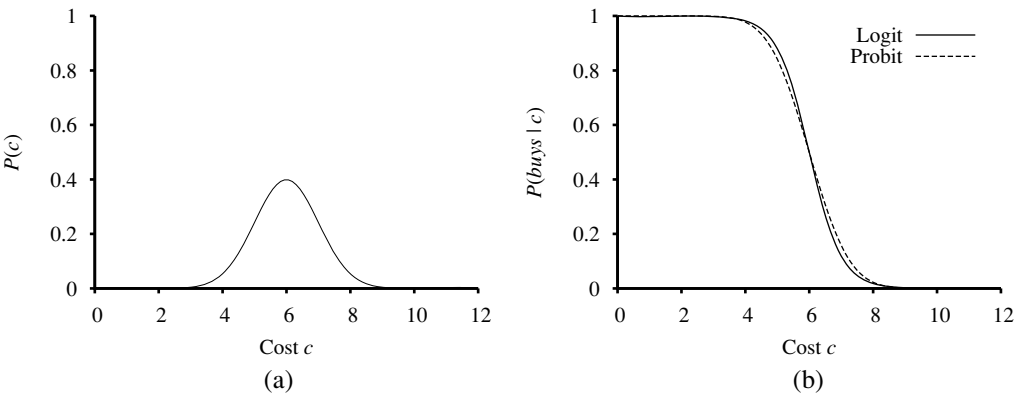


Figure 14.7 (a) A normal (Gaussian) distribution for the cost threshold, centered on $\mu = 6.0$ with standard deviation $\sigma = 1.0$. (b) Logit and probit distributions for the probability of $buys$ given $cost$, for the parameters $\mu = 6.0$ and $\sigma = 1.0$.

PROBIT DISTRIBUTION

bution (pronounced "pro-bit" and short for "probability unit") is illustrated in Figure 14.7(a). The form can be justified by proposing that the underlying decision process has a hard threshold, but that the precise location of the threshold is subject to random Gaussian noise.

LOGIT DISTRIBUTION

LOGISTIC FUNCTION

An alternative to the probit model is the **logit distribution** (pronounced "low-jit"). It uses the **logistic function** $1/(1 + e^{-x})$ to produce a soft threshold:

$$P(buys \mid Cost = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})} .$$

This is illustrated in Figure 14.7(b). The two distributions look similar, but the logit actually has much longer "tails." The probit is often a better fit to real situations, but the logit is sometimes easier to deal with mathematically. It is used widely in neural networks (Chapter 20). Both probit and logit can be generalized to handle multiple continuous parents by taking a linear combination of the parent values.

14.4 EXACT INFERENCE IN BAYESIAN NETWORKS

EVENT

HIDDEN VARIABLE

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of **query variables**, given some observed **event**—that is, some assignment of values to a set of **evidence variables**. To simplify the presentation, we will consider only one query variable at a time; the algorithms can easily be extended to queries with multiple variables. We will use the notation from Chapter 13: X denotes the query variable; $\mathbf{E}$ denotes the set of evidence variables $E_1, \ldots, E_m$, and $\mathbf{e}$ is a particular observed event; $\mathbf{Y}$ will denotes the nonevidence, nonquery variables $Y_1, \ldots, Y_l$ (called the **hidden variables**). Thus, the complete set of variables is $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. A typical query asks for the posterior probability distribution $\mathbf{P}(X \mid \mathbf{e})$.

In the burglary network, we might observe the event in which $JohnCalls = true$ and $MaryCalls = true$. We could then ask for, say, the probability that a burglary has occurred:

$$\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true) = \langle 0.284, 0.716 \rangle \ .$$

In this section we discuss exact algorithms for computing posterior probabilities and will consider the complexity of this task. It turns out that the general case is intractable, so Section 14.5 covers methods for approximate inference.

14.4.1 Inference by enumeration

Chapter 13 explained that any conditional probability can be computed by summing terms from the full joint distribution. More specifically, a query $\mathbf{P}(X \mid \mathbf{e})$ can be answered using Equation (13.9), which we repeat here for convenience:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \ .$$

Now, a Bayesian network gives a complete representation of the full joint distribution. More specifically, Equation (14.2) on page 513 shows that the terms $P(x, \mathbf{e}, \mathbf{y})$ in the joint distribution can be written as products of conditional probabilities from the network. Therefore, *a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.*

Consider the query $\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true)$. The hidden variables for this query are $Earthquake$ and $Alarm$. From Equation (13.9), using initial letters for the variables to shorten the expressions, we have[4]

$$\mathbf{P}(B \mid j, m) = \alpha \, \mathbf{P}(B, j, m) = \alpha \sum_{e} \sum_{a} \mathbf{P}(B, j, m, e, a,) \ .$$

The semantics of Bayesian networks (Equation (14.2)) then gives us an expression in terms of CPT entries. For simplicity, we do this just for $Burglary = true$:

$$P(b \mid j, m) = \alpha \sum_{e} \sum_{a} P(b) P(e) P(a \mid b, e) P(j \mid a) P(m \mid a) \ .$$

To compute this expression, we have to add four terms, each computed by multiplying five numbers. In the worst case, where we have to sum out almost all the variables, the complexity of the algorithm for a network with n Boolean variables is $O(n2^n)$.

An improvement can be obtained from the following simple observations: the $P(b)$ term is a constant and can be moved outside the summations over a and e, and the $P(e)$ term can be moved outside the summation over a. Hence, we have

$$P(b \mid j, m) = \alpha \, P(b) \sum_{e} P(e) \sum_{a} P(a \mid b, e) P(j \mid a) P(m \mid a) \ . \tag{14.4}$$

This expression can be evaluated by looping through the variables in order, multiplying CPT entries as we go. For each summation, we also need to loop over the variable's possible

[4] An expression such as $\sum_e P(a, e)$ means to sum $P(A = a, E = e)$ for all possible values of e. When E is Boolean, there is an ambiguity in that $P(e)$ is used to mean both $P(E = true)$ and $P(E = e)$, but it should be clear from context which is intended; in particular, in the context of a sum the latter is intended.

values. The structure of this computation is shown in Figure 14.8. Using the numbers from Figure 14.2, we obtain $P(b \mid j, m) = \alpha \times 0.00059224$. The corresponding computation for $\neg b$ yields $\alpha \times 0.0014919$; hence,

$$\mathbf{P}(B \mid j, m) = \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle .$$

That is, the chance of a burglary, given calls from both neighbors, is about 28%.

The evaluation process for the expression in Equation (14.4) is shown as an expression tree in Figure 14.8. The ENUMERATION-ASK algorithm in Figure 14.9 evaluates such trees using depth-first recursion. The algorithm is very similar in structure to the backtracking algorithm for solving CSPs (Figure 6.5) and the DPLL algorithm for satisfiability (Figure 7.17).

The space complexity of ENUMERATION-ASK is only linear in the number of variables: the algorithm sums over the full joint distribution without ever constructing it explicitly. Unfortunately, its time complexity for a network with n Boolean variables is always $O(2^n)$—better than the $O(n\,2^n)$ for the simple approach described earlier, but still rather grim.

Note that the tree in Figure 14.8 makes explicit the *repeated subexpressions* evaluated by the algorithm. The products $P(j \mid a)P(m \mid a)$ and $P(j \mid \neg a)P(m \mid \neg a)$ are computed twice, once for each value of e. The next section describes a general method that avoids such wasted computations.

14.4.2 The variable elimination algorithm

The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in Figure 14.8. The idea is simple: do the calculation once and save the results for later use. This is a form of dynamic programming. There are several versions of this approach; we present the **variable elimination** algorithm, which is the simplest. Variable elimination works by evaluating expressions such as Equation (14.4) in *right-to-left* order (that is, *bottom up* in Figure 14.8). Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.

VARIABLE ELIMINATION

Let us illustrate this process for the burglary network. We evaluate the expression

$$\mathbf{P}(B \mid j, m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_e \underbrace{P(e)}_{\mathbf{f}_2(E)} \sum_a \underbrace{\mathbf{P}(a \mid B, e)}_{\mathbf{f}_3(A,B,E)} \underbrace{P(j \mid a)}_{\mathbf{f}_4(A)} \underbrace{P(m \mid a)}_{\mathbf{f}_5(A)} .$$

Notice that we have annotated each part of the expression with the name of the corresponding **factor**; each factor is a matrix indexed by the values of its argument variables. For example, the factors $\mathbf{f}_4(A)$ and $\mathbf{f}_5(A)$ corresponding to $P(j \mid a)$ and $P(m \mid a)$ depend just on A because J and M are fixed by the query. They are therefore two-element vectors:

FACTOR

$$\mathbf{f}_4(A) = \begin{pmatrix} P(j \mid a) \\ P(j \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix} \qquad \mathbf{f}_5(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.01 \end{pmatrix} .$$

$\mathbf{f}_3(A, B, E)$ will be a $2 \times 2 \times 2$ matrix, which is hard to show on the printed page. (The "first" element is given by $P(a \mid b, e) = 0.95$ and the "last" by $P(\neg a \mid \neg b, \neg e) = 0.999$.) In terms of factors, the query expression is written as

$$\mathbf{P}(B \mid j, m) = \alpha \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$

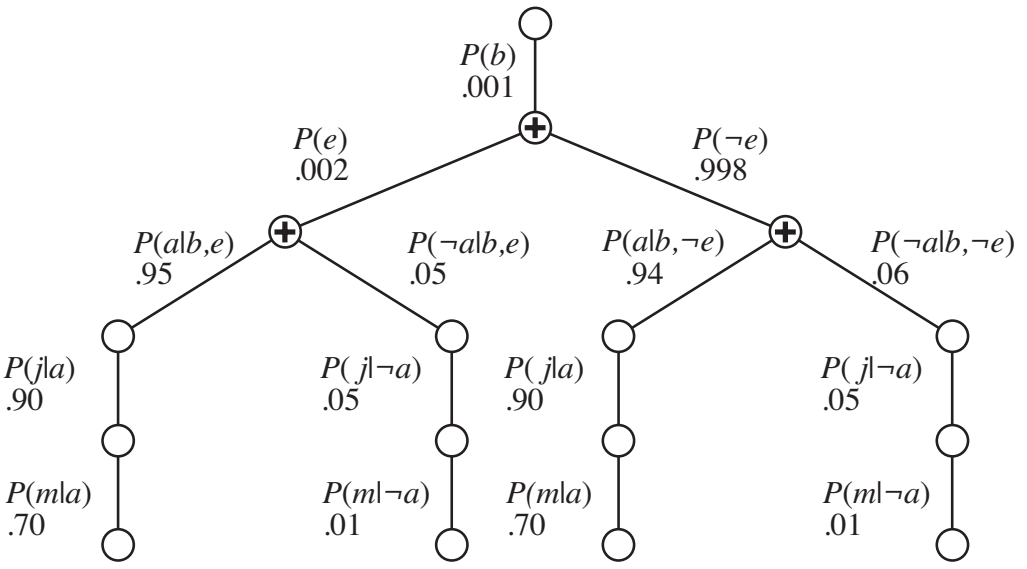


Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

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 $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ /* **Y** = *hidden variables* */

  $\mathbf{Q}(X) \leftarrow$ a distribution over X, initially empty
  for each value x_i of X **do**
    $\mathbf{Q}(x_i) \leftarrow$ ENUMERATE-ALL(bn.VARS, $\mathbf{e}_{x_i}$)
      where $\mathbf{e}_{x_i}$ is **e** extended with $X = x_i$
  return NORMALIZE($\mathbf{Q}(X)$)

---

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 $P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), **e**)
    else return $\sum_y P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_y$)
      where $\mathbf{e}_y$ is **e** extended with $Y = y$

Figure 14.9 The enumeration algorithm for answering queries on Bayesian networks.

where the "$\times$" operator is not ordinary matrix multiplication but instead the **pointwise product** operation, to be described shortly.

POINTWISE PRODUCT

The process of evaluation is a process of summing out variables (right to left) from pointwise products of factors to produce new factors, eventually yielding a factor that is the solution, i.e., the posterior distribution over the query variable. The steps are as follows:

- First, we sum out A from the product of $\mathbf{f}_3$, $\mathbf{f}_4$, and $\mathbf{f}_5$. This gives us a new 2×2 factor $\mathbf{f}_6(B, E)$ whose indices range over just B and E:

$$\begin{aligned}\mathbf{f}_6(B, E) &= \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) \\ &= (\mathbf{f}_3(a, B, E) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a)) + (\mathbf{f}_3(\neg a, B, E) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a)) \,.\end{aligned}$$

  Now we are left with the expression

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \,.$$

- Next, we sum out E from the product of $\mathbf{f}_2$ and $\mathbf{f}_6$:

$$\begin{aligned}\mathbf{f}_7(B) &= \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \\ &= \mathbf{f}_2(e) \times \mathbf{f}_6(B, e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B, \neg e) \,.\end{aligned}$$

  This leaves the expression

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

  which can be evaluated by taking the pointwise product and normalizing the result.

Examining this sequence, we see that two basic computational operations are required: pointwise product of a pair of factors, and summing out a variable from a product of factors. The next section describes each of these operations.

Operations on factors

The pointwise product of two factors $\mathbf{f}_1$ and $\mathbf{f}_2$ yields a new factor $\mathbf{f}$ whose variables are the *union* of the variables in $\mathbf{f}_1$ and $\mathbf{f}_2$ and whose elements are given by the product of the corresponding elements in the two factors. Suppose the two factors have variables $Y_1, \ldots, Y_k$ in common. Then we have

$$\mathbf{f}(X_1 \ldots X_j, Y_1 \ldots Y_k, Z_1 \ldots Z_l) = \mathbf{f}_1(X_1 \ldots X_j, Y_1 \ldots Y_k)\, \mathbf{f}_2(Y_1 \ldots Y_k, Z, \ldots Z_l).$$

If all the variables are binary, then $\mathbf{f}_1$ and $\mathbf{f}_2$ have 2^{j+k} and 2^{k+l} entries, respectively, and the pointwise product has 2^{j+k+l} entries. For example, given two factors $\mathbf{f}_1(A, B)$ and $\mathbf{f}_2(B, C)$, the pointwise product $\mathbf{f}_1 \times \mathbf{f}_2 = \mathbf{f}_3(A, B, C)$ has $2^{1+1+1} = 8$ entries, as illustrated in Figure 14.10. Notice that the factor resulting from a pointwise product can contain more variables than any of the factors being multiplied and that the size of a factor is exponential in the number of variables. This is where both space and time complexity arise in the variable elimination algorithm.

A	B	$\mathbf{f}_1(A,B)$	B	C	$\mathbf{f}_2(B,C)$	A	B	C	$\mathbf{f}_3(A,B,C)$
T	T	.3	T	T	.2	T	T	T	$.3 \times .2 = .06$
T	F	.7	T	F	.8	T	T	F	$.3 \times .8 = .24$
F	T	.9	F	T	.6	T	F	T	$.7 \times .6 = .42$
F	F	.1	F	F	.4	T	F	F	$.7 \times .4 = .28$
						F	T	T	$.9 \times .2 = .18$
						F	T	F	$.9 \times .8 = .72$
						F	F	T	$.1 \times .6 = .06$
						F	F	F	$.1 \times .4 = .04$

Figure 14.10 Illustrating pointwise multiplication: $\mathbf{f}_1(A,B) \times \mathbf{f}_2(B,C) = \mathbf{f}_3(A,B,C)$.

Summing out a variable from a product of factors is done by adding up the submatrices formed by fixing the variable to each of its values in turn. For example, to sum out A from $\mathbf{f}_3(A,B,C)$, we write

$$\begin{aligned}\mathbf{f}(B,C) &= \sum_a \mathbf{f}_3(A,B,C) = \mathbf{f}_3(a,B,C) + \mathbf{f}_3(\neg a,B,C) \\ &= \begin{pmatrix} .06 & .24 \\ .42 & .28 \end{pmatrix} + \begin{pmatrix} .18 & .72 \\ .06 & .04 \end{pmatrix} = \begin{pmatrix} .24 & .96 \\ .48 & .32 \end{pmatrix} .\end{aligned}$$

The only trick is to notice that any factor that does *not* depend on the variable to be summed out can be moved outside the summation. For example, if we were to sum out E first in the burglary network, the relevant part of the expression would be

$$\sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) = \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) .$$

Now the pointwise product inside the summation is computed, and the variable is summed out of the resulting matrix.

Notice that matrices are *not* multiplied until we need to sum out a variable from the accumulated product. At that point, we multiply just those matrices that include the variable to be summed out. Given functions for pointwise product and summing out, the variable elimination algorithm itself can be written quite simply, as shown in Figure 14.11.

Variable ordering and variable relevance

The algorithm in Figure 14.11 includes an unspecified ORDER function to choose an ordering for the variables. Every choice of ordering yields a valid algorithm, but different orderings cause different intermediate factors to be generated during the calculation. For example, in the calculation shown previously, we eliminated A before E; if we do it the other way, the calculation becomes

$$\mathbf{P}(B \mid j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) ,$$

during which a new factor $\mathbf{f}_6(A,B)$ will be generated.

In general, the time and space requirements of variable elimination are dominated by the size of the largest factor constructed during the operation of the algorithm. This in turn

```
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, ..., X_n)

  factors ← [ ]
  for each var in ORDER(bn.VARS) do
      factors ← [MAKE-FACTOR(var, e)|factors]
      if var is a hidden variable then factors ← SUM-OUT(var, factors)
  return NORMALIZE(POINTWISE-PRODUCT(factors))
```

Figure 14.11 The variable elimination algorithm for inference in Bayesian networks.

is determined by the order of elimination of variables and by the structure of the network. It turns out to be intractable to determine the optimal ordering, but several good heuristics are available. One fairly effective method is a greedy one: eliminate whichever variable minimizes the size of the next factor to be constructed.

Let us consider one more query: $\mathbf{P}(JohnCalls \mid Burglary = true)$. As usual, the first step is to write out the nested summation:

$$\mathbf{P}(J \mid b) = \alpha\, P(b) \sum_e P(e) \sum_a P(a \mid b, e) \mathbf{P}(J \mid a) \sum_m P(m \mid a)\ .$$

Evaluating this expression from right to left, we notice something interesting: $\sum_m P(m \mid a)$ is equal to 1 by definition! Hence, there was no need to include it in the first place; the variable M is *irrelevant* to this query. Another way of saying this is that the result of the query $P(JohnCalls \mid Burglary = true)$ is unchanged if we remove $MaryCalls$ from the network altogether. In general, we can remove any leaf node that is not a query variable or an evidence variable. After its removal, there may be some more leaf nodes, and these too may be irrelevant. Continuing this process, we eventually find that *every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.* A variable elimination algorithm can therefore remove all these variables before evaluating the query.

14.4.3 The complexity of exact inference

The complexity of exact inference in Bayesian networks depends strongly on the structure of the network. The burglary network of Figure 14.2 belongs to the family of networks in which there is at most one undirected path between any two nodes in the network. These are called **singly connected** networks or **polytrees**, and they have a particularly nice property: *The time and space complexity of exact inference in polytrees is linear in the size of the network.* Here, the size is defined as the number of CPT entries; if the number of parents of each node is bounded by a constant, then the complexity will also be linear in the number of nodes.

SINGLY CONNECTED

POLYTREE

MULTIPLY CONNECTED

For **multiply connected** networks, such as that of Figure 14.12(a), variable elimination can have exponential time and space complexity in the worst case, even when the number of parents per node is bounded. This is not surprising when one considers that *because it*

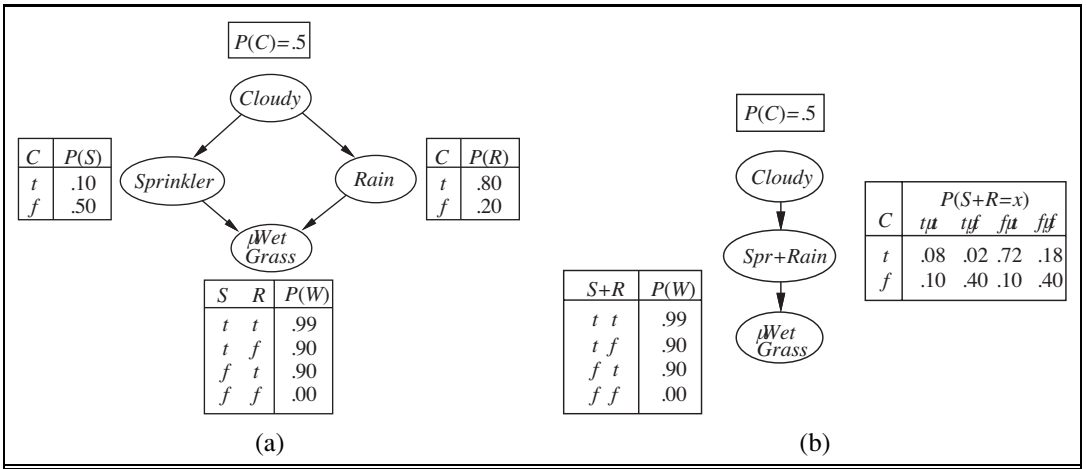


Figure 14.12 (a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

includes inference in propositional logic as a special case, inference in Bayesian networks is NP-hard. In fact, it can be shown (Exercise 14.16) that the problem is as hard as that of computing the *number* of satisfying assignments for a propositional logic formula. This means that it is #P-hard ("number-P hard")—that is, strictly harder than NP-complete problems.

There is a close connection between the complexity of Bayesian network inference and the complexity of constraint satisfaction problems (CSPs). As we discussed in Chapter 6, the difficulty of solving a discrete CSP is related to how "treelike" its constraint graph is. Measures such as **tree width**, which bound the complexity of solving a CSP, can also be applied directly to Bayesian networks. Moreover, the variable elimination algorithm can be generalized to solve CSPs as well as Bayesian networks.

14.4.4 Clustering algorithms

The variable elimination algorithm is simple and efficient for answering individual queries. If we want to compute posterior probabilities for all the variables in a network, however, it can be less efficient. For example, in a polytree network, one would need to issue $O(n)$ queries costing $O(n)$ each, for a total of $O(n^2)$ time. Using **clustering** algorithms (also known as **join tree** algorithms), the time can be reduced to $O(n)$. For this reason, these algorithms are widely used in commercial Bayesian network tools.

CLUSTERING

JOIN TREE

The basic idea of clustering is to join individual nodes of the network to form cluster nodes in such a way that the resulting network is a polytree. For example, the multiply connected network shown in Figure 14.12(a) can be converted into a polytree by combining the $Sprinkler$ and $Rain$ node into a cluster node called $Sprinkler{+}Rain$, as shown in Figure 14.12(b). The two Boolean nodes are replaced by a "meganode" that takes on four possible values: tt, tf, ft, and ff. The meganode has only one parent, the Boolean variable $Cloudy$, so there are two conditioning cases. Although this example doesn't show it, the process of clustering often produces meganodes that share some variables.

Once the network is in polytree form, a special-purpose inference algorithm is required, because ordinary inference methods cannot handle meganodes that share variables with each other. Essentially, the algorithm is a form of constraint propagation (see Chapter 6) where the constraints ensure that neighboring meganodes agree on the posterior probability of any variables that they have in common. With careful bookkeeping, this algorithm is able to compute posterior probabilities for all the nonevidence nodes in the network in time *linear* in the size of the clustered network. However, the NP-hardness of the problem has not disappeared: if a network requires exponential time and space with variable elimination, then the CPTs in the clustered network will necessarily be exponentially large.

14.5 Approximate Inference in Bayesian Networks

Given the intractability of exact inference in large, multiply connected networks, it is essential to consider approximate inference methods. This section describes randomized sampling algorithms, also called **Monte Carlo** algorithms, that provide approximate answers whose accuracy depends on the number of samples generated. Monte Carlo algorithms, of which simulated annealing (page 126) is an example, are used in many branches of science to estimate quantities that are difficult to calculate exactly. In this section, we are interested in sampling applied to the computation of posterior probabilities. We describe two families of algorithms: direct sampling and Markov chain sampling. Two other approaches—variational methods and loopy propagation—are mentioned in the notes at the end of the chapter.

MONTE CARLO

14.5.1 Direct sampling methods

The primitive element in any sampling algorithm is the generation of samples from a known probability distribution. For example, an unbiased coin can be thought of as a random variable $Coin$ with values $\langle heads, tails \rangle$ and a prior distribution $\mathbf{P}(Coin) = \langle 0.5, 0.5 \rangle$. Sampling from this distribution is exactly like flipping the coin: with probability 0.5 it will return $heads$, and with probability 0.5 it will return $tails$. Given a source of random numbers uniformly distributed in the range $[0, 1]$, it is a simple matter to sample any distribution on a single variable, whether discrete or continuous. (See Exercise 14.17.)

The simplest kind of random sampling process for Bayesian networks generates events from a network that has no evidence associated with it. The idea is to sample each variable in turn, in topological order. The probability distribution from which the value is sampled is conditioned on the values already assigned to the variable's parents. This algorithm is shown in Figure 14.13. We can illustrate its operation on the network in Figure 14.12(a), assuming an ordering $[Cloudy, Sprinkler, Rain, WetGrass]$:

1. Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$, value is $true$.
2. Sample from $\mathbf{P}(Sprinkler \mid Cloudy = true) = \langle 0.1, 0.9 \rangle$, value is $false$.
3. Sample from $\mathbf{P}(Rain \mid Cloudy = true) = \langle 0.8, 0.2 \rangle$, value is $true$.
4. Sample from $\mathbf{P}(WetGrass \mid Sprinkler = false, Rain = true) = \langle 0.9, 0.1 \rangle$, value is $true$.

In this case, PRIOR-SAMPLE returns the event $[true, false, true, true]$.

function PRIOR-SAMPLE(bn) **returns** an event sampled from the prior specified by bn
  inputs: bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with n elements
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return x

Figure 14.13 A sampling algorithm that generates events from a Bayesian network. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents.

It is easy to see that PRIOR-SAMPLE generates samples from the prior joint distribution specified by the network. First, let $S_{PS}(x_1, \ldots, x_n)$ be the probability that a specific event is generated by the PRIOR-SAMPLE algorithm. *Just looking at the sampling process*, we have

$$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i \,|\, parents(X_i))$$

because each sampling step depends only on the parent values. This expression should look familiar, because it is also the probability of the event according to the Bayesian net's representation of the joint distribution, as stated in Equation (14.2). That is, we have

$$S_{PS}(x_1 \ldots x_n) = P(x_1 \ldots x_n) \; .$$

This simple fact makes it easy to answer questions by using samples.

In any sampling algorithm, the answers are computed by counting the actual samples generated. Suppose there are N total samples, and let $N_{PS}(x_1, \ldots, x_n)$ be the number of times the specific event $x_1, \ldots, x_n$ occurs in the set of samples. We expect this number, as a fraction of the total, to converge in the limit to its expected value according to the sampling probability:

$$\lim_{N \to \infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) \; . \tag{14.5}$$

For example, consider the event produced earlier: $[true, false, true, true]$. The sampling probability for this event is

$$S_{PS}(true, false, true, true) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 \; .$$

Hence, in the limit of large N, we expect 32.4% of the samples to be of this event.

Whenever we use an approximate equality ("$\approx$") in what follows, we mean it in exactly this sense—that the estimated probability becomes exact in the large-sample limit. Such an estimate is called **consistent**. For example, one can produce a consistent estimate of the probability of any partially specified event $x_1, \ldots, x_m$, where $m \leq n$, as follows:

CONSISTENT

$$P(x_1, \ldots, x_m) \approx N_{PS}(x_1, \ldots, x_m)/N \; . \tag{14.6}$$

That is, the probability of the event can be estimated as the fraction of all complete events generated by the sampling process that match the partially specified event. For example, if

we generate 1000 samples from the sprinkler network, and 511 of them have $Rain = true$, then the estimated probability of rain, written as $\hat{P}(Rain = true)$, is 0.511.

Rejection sampling in Bayesian networks

REJECTION SAMPLING

Rejection sampling is a general method for producing samples from a hard-to-sample distribution given an easy-to-sample distribution. In its simplest form, it can be used to compute conditional probabilities—that is, to determine $P(X \mid \mathbf{e})$. The REJECTION-SAMPLING algorithm is shown in Figure 14.14. First, it generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate $\hat{P}(X = x \mid \mathbf{e})$ is obtained by counting how often $X = x$ occurs in the remaining samples.

Let $\hat{\mathbf{P}}(X \mid \mathbf{e})$ be the estimated distribution that the algorithm returns. From the definition of the algorithm, we have

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e}) = \frac{\mathbf{N}_{PS}(X, \mathbf{e})}{N_{PS}(\mathbf{e})} .$$

From Equation (14.6), this becomes

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X \mid \mathbf{e}) .$$

That is, rejection sampling produces a consistent estimate of the true probability.

Continuing with our example from Figure 14.12(a), let us assume that we wish to estimate $\mathbf{P}(Rain \mid Sprinkler = true)$, using 100 samples. Of the 100 that we generate, suppose that 73 have $Sprinkler = false$ and are rejected, while 27 have $Sprinkler = true$; of the 27, 8 have $Rain = true$ and 19 have $Rain = false$. Hence,

$$\mathbf{P}(Rain \mid Sprinkler = true) \approx \text{NORMALIZE}(\langle 8, 19 \rangle) = \langle 0.296, 0.704 \rangle .$$

The true answer is $\langle 0.3, 0.7 \rangle$. As more samples are collected, the estimate will converge to the true answer. The standard deviation of the error in each probability will be proportional to $1/\sqrt{n}$, where n is the number of samples used in the estimate.

The biggest problem with rejection sampling is that it rejects so many samples! The fraction of samples consistent with the evidence $\mathbf{e}$ drops exponentially as the number of evidence variables grows, so the procedure is simply unusable for complex problems.

Notice that rejection sampling is very similar to the estimation of conditional probabilities directly from the real world. For example, to estimate $\mathbf{P}(Rain \mid RedSkyAtNight = true)$, one can simply count how often it rains after a red sky is observed the previous evening—ignoring those evenings when the sky is not red. (Here, the world itself plays the role of the sample-generation algorithm.) Obviously, this could take a long time if the sky is very seldom red, and that is the weakness of rejection sampling.

Likelihood weighting

LIKELIHOOD WEIGHTING

IMPORTANCE SAMPLING

Likelihood weighting avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence $\mathbf{e}$. It is a particular instance of the general statistical technique of **importance sampling**, tailored for inference in Bayesian networks. We begin by

function REJECTION-SAMPLING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  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 $\leftarrow$ 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**)

Figure 14.14 The rejection-sampling algorithm for answering queries given evidence in a Bayesian network.

describing how the algorithm works; then we show that it works correctly—that is, generates consistent probability estimates.

LIKELIHOOD-WEIGHTING (see Figure 14.15) fixes the values for the evidence variables **E** and samples only the nonevidence variables. This guarantees that each event generated is consistent with the evidence. Not all events are equal, however. Before tallying the counts in the distribution for the query variable, each event is weighted by the *likelihood* that the event accords to the evidence, as measured by the product of the conditional probabilities for each evidence variable, given its parents. Intuitively, events in which the actual evidence appears unlikely should be given less weight.

Let us apply the algorithm to the network shown in Figure 14.12(a), with the query $\mathbf{P}(Rain \mid Cloudy = true, WetGrass = true)$ and the ordering *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*. (Any topological ordering will do.) The process goes as follows: First, the weight w is set to 1.0. Then an event is generated:

1. $Cloudy$ is an evidence variable with value $true$. Therefore, we set
$$w \leftarrow w \times P(Cloudy = true) = 0.5 \ .$$
2. $Sprinkler$ is not an evidence variable, so sample from $\mathbf{P}(Sprinkler \mid Cloudy = true) = \langle 0.1, 0.9 \rangle$; suppose this returns $false$.
3. Similarly, sample from $\mathbf{P}(Rain \mid Cloudy = true) = \langle 0.8, 0.2 \rangle$; suppose this returns $true$.
4. $WetGrass$ is an evidence variable with value $true$. Therefore, we set
$$w \leftarrow w \times P(WetGrass = true \mid Sprinkler = false, Rain = true) = 0.45 \ .$$

Here WEIGHTED-SAMPLE returns the event $[true, false, true, true]$ with weight 0.45, and this is tallied under $Rain = true$.

To understand why likelihood weighting works, we start by examining the sampling probability S_{WS} for WEIGHTED-SAMPLE. Remember that the evidence variables **E** are fixed

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
    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, $w \leftarrow$ WEIGHTED-SAMPLE(bn, **e**)
    $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ 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** $X_1, \ldots, X_n$ **do**
    if X_i is an evidence variable with value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else **x**$[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return **x**, w

Figure 14.15 The likelihood-weighting algorithm for inference in Bayesian networks. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

with values **e**. We call the nonevidence variables **Z** (including the query variable X). The algorithm samples each variable in **Z** given its parent values:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \ . \tag{14.7}$$

Notice that $Parents(Z_i)$ can include both nonevidence variables and evidence variables. Unlike the prior distribution $P(\mathbf{z})$, the distribution S_{WS} pays some attention to the evidence: the sampled values for each Z_i will be influenced by evidence among Z_i's ancestors. For example, when sampling $Sprinkler$ the algorithm pays attention to the evidence $Cloudy = true$ in its parent variable. On the other hand, S_{WS} pays less attention to the evidence than does the true posterior distribution $P(\mathbf{z} \mid \mathbf{e})$, because the sampled values for each Z_i *ignore* evidence among Z_i's non-ancestors.[5] For example, when sampling $Sprinkler$ and $Rain$ the algorithm ignores the evidence in the child variable $WetGrass = true$; this means it will generate many samples with $Sprinkler = false$ and $Rain = false$ despite the fact that the evidence actually rules out this case.

[5] Ideally, we would like to use a sampling distribution equal to the true posterior $P(\mathbf{z} \mid \mathbf{e})$, to take all the evidence into account. This cannot be done efficiently, however. If it could, then we could approximate the desired probability to arbitrary accuracy with a polynomial number of samples. It can be shown that no such polynomial-time approximation scheme can exist.

The likelihood weight w makes up for the difference between the actual and desired sampling distributions. The weight for a given sample $\mathbf{x}$, composed from $\mathbf{z}$ and $\mathbf{e}$, is the product of the likelihoods for each evidence variable given its parents (some or all of which may be among the Z_is):

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i \,|\, parents(E_i)) \; . \tag{14.8}$$

Multiplying Equations (14.7) and (14.8), we see that the *weighted* probability of a sample has the particularly convenient form

$$\begin{aligned} S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e}) \; &= \; \prod_{i=1}^{l} P(z_i \,|\, parents(Z_i)) \prod_{i=1}^{m} P(e_i \,|\, parents(E_i)) \\ &= \; P(\mathbf{z}, \mathbf{e}) \end{aligned} \tag{14.9}$$

because the two products cover all the variables in the network, allowing us to use Equation (14.2) for the joint probability.

Now it is easy to show that likelihood weighting estimates are consistent. For any particular value x of X, the estimated posterior probability can be calculated as follows:

$$\begin{aligned} \hat{P}(x \,|\, \mathbf{e}) \; &= \; \alpha \sum_{\mathbf{y}} N_{WS}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) && \text{from LIKELIHOOD-WEIGHTING} \\ &\approx \; \alpha' \sum_{\mathbf{y}} S_{WS}(x, \mathbf{y}, \mathbf{e}) w(x, \mathbf{y}, \mathbf{e}) && \text{for large } N \\ &= \; \alpha' \sum_{\mathbf{y}} P(x, \mathbf{y}, \mathbf{e}) && \text{by Equation (14.9)} \\ &= \; \alpha' P(x, \mathbf{e}) = P(x \,|\, \mathbf{e}) \; . \end{aligned}$$

Hence, likelihood weighting returns consistent estimates.

Because likelihood weighting uses all the samples generated, it can be much more efficient than rejection sampling. It will, however, suffer a degradation in performance as the number of evidence variables increases. This is because most samples will have very low weights and hence the weighted estimate will be dominated by the tiny fraction of samples that accord more than an infinitesimal likelihood to the evidence. The problem is exacerbated if the evidence variables occur late in the variable ordering, because then the nonevidence variables will have no evidence in their parents and ancestors to guide the generation of samples. This means the samples will be simulations that bear little resemblance to the reality suggested by the evidence.

14.5.2 Inference by Markov chain simulation

MARKOV CHAIN MONTE CARLO

Markov chain Monte Carlo (MCMC) algorithms work quite differently from rejection sampling and likelihood weighting. Instead of generating each sample from scratch, MCMC algorithms generate each sample by making a random change to the preceding sample. It is therefore helpful to think of an MCMC algorithm as being in a particular *current state* specifying a value for every variable and generating a *next state* by making random changes to the

current state. (If this reminds you of simulated annealing from Chapter 4 or WALKSAT from Chapter 7, that is because both are members of the MCMC family.) Here we describe a particular form of MCMC called **Gibbs sampling**, which is especially well suited for Bayesian networks. (Other forms, some of them significantly more powerful, are discussed in the notes at the end of the chapter.) We will first describe what the algorithm does, then we will explain why it works.

GIBBS SAMPLING

Gibbs sampling in Bayesian networks

The Gibbs sampling algorithm for Bayesian networks starts with an arbitrary state (with the evidence variables fixed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables X_i. The sampling for X_i is done *conditioned on the current values of the variables in the Markov blanket of* X_i. (Recall from page 517 that the Markov blanket of a variable consists of its parents, children, and children's parents.) The algorithm therefore wanders randomly around the state space—the space of possible complete assignments—flipping one variable at a time, but keeping the evidence variables fixed.

Consider the query $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true)$ applied to the network in Figure 14.12(a). The evidence variables $Sprinkler$ and $WetGrass$ are fixed to their observed values and the nonevidence variables $Cloudy$ and $Rain$ are initialized randomly—let us say to $true$ and $false$ respectively. Thus, the initial state is $[true, true, false, true]$. Now the nonevidence variables are sampled repeatedly in an arbitrary order. For example:

1. $Cloudy$ is sampled, given the current values of its Markov blanket variables: in this case, we sample from $\mathbf{P}(Cloudy \mid Sprinkler = true, Rain = false)$. (Shortly, we will show how to calculate this distribution.) Suppose the result is $Cloudy = false$. Then the new current state is $[false, true, false, true]$.
2. $Rain$ is sampled, given the current values of its Markov blanket variables: in this case, we sample from $\mathbf{P}(Rain \mid Cloudy = false, Sprinkler = true, WetGrass = true)$. Suppose this yields $Rain = true$. The new current state is $[false, true, true, true]$.

Each state visited during this process is a sample that contributes to the estimate for the query variable $Rain$. If the process visits 20 states where $Rain$ is true and 60 states where $Rain$ is false, then the answer to the query is $\text{NORMALIZE}(\langle 20, 60 \rangle) = \langle 0.25, 0.75 \rangle$. The complete algorithm is shown in Figure 14.16.

Why Gibbs sampling works

We will now show that Gibbs sampling returns consistent estimates for posterior probabilities. The material in this section is quite technical, but the basic claim is straightforward: *the sampling process settles into a "dynamic equilibrium" in which the long-run fraction of time spent in each state is exactly proportional to its posterior probability.* This remarkable property follows from the specific **transition probability** with which the process moves from one state to another, as defined by the conditional distribution given the Markov blanket of the variable being sampled.

TRANSITION PROBABILITY

```
function GIBBS-ASK(X, e, bn, N) returns an estimate of P(X|e)
  local variables: N, a vector of counts for each value of X, initially zero
                   Z, the nonevidence variables in bn
                   x, the current state of the network, initially copied from e

  initialize x with random values for the variables in Z
  for j = 1 to N do
    for each Z_i in Z do
      set the value of Z_i in x by sampling from P(Z_i|mb(Z_i))
      N[x] ← N[x] + 1 where x is the value of X in x
  return NORMALIZE(N)
```

Figure 14.16 The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.

MARKOV CHAIN

Let $q(\mathbf{x} \rightarrow \mathbf{x}')$ be the probability that the process makes a transition from state $\mathbf{x}$ to state $\mathbf{x}'$. This transition probability defines what is called a **Markov chain** on the state space. (Markov chains also figure prominently in Chapters 15 and 17.) Now suppose that we run the Markov chain for t steps, and let $\pi_t(\mathbf{x})$ be the probability that the system is in state $\mathbf{x}$ at time t. Similarly, let $\pi_{t+1}(\mathbf{x}')$ be the probability of being in state $\mathbf{x}'$ at time $t+1$. Given $\pi_t(\mathbf{x})$, we can calculate $\pi_{t+1}(\mathbf{x}')$ by summing, for all states the system could be in at time t, the probability of being in that state times the probability of making the transition to $\mathbf{x}'$:

$$\pi_{t+1}(\mathbf{x}') = \sum_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \ .$$

STATIONARY DISTRIBUTION

We say that the chain has reached its **stationary distribution** if $\pi_t = \pi_{t+1}$. Let us call this stationary distribution π; its defining equation is therefore

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \qquad \text{for all } \mathbf{x}' \ . \tag{14.10}$$

ERGODIC

Provided the transition probability distribution q is **ergodic**—that is, every state is reachable from every other and there are no strictly periodic cycles—there is exactly one distribution π satisfying this equation for any given q.

Equation (14.10) can be read as saying that the expected "outflow" from each state (i.e., its current "population") is equal to the expected "inflow" from all the states. One obvious way to satisfy this relationship is if the expected flow between any pair of states is the same in both directions; that is,

$$\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) \qquad \text{for all } \mathbf{x}, \ \mathbf{x}' \ . \tag{14.11}$$

DETAILED BALANCE

When these equations hold, we say that $q(\mathbf{x} \rightarrow \mathbf{x}')$ is in **detailed balance** with $\pi(\mathbf{x})$.

We can show that detailed balance implies stationarity simply by summing over $\mathbf{x}$ in Equation (14.11). We have

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}') \sum_{\mathbf{x}} q(\mathbf{x}' \rightarrow \mathbf{x}) = \pi(\mathbf{x}')$$

where the last step follows because a transition from $\mathbf{x}'$ is guaranteed to occur.

The transition probability $q(\mathbf{x} \rightarrow \mathbf{x}')$ defined by the sampling step in GIBBS-ASK is actually a special case of the more general definition of Gibbs sampling, according to which each variable is sampled conditionally on the current values of *all* the other variables. We start by showing that this general definition of Gibbs sampling satisfies the detailed balance equation with a stationary distribution equal to $P(\mathbf{x} \,|\, \mathbf{e})$, (the true posterior distribution on the nonevidence variables). Then, we simply observe that, for Bayesian networks, sampling conditionally on all variables is equivalent to sampling conditionally on the variable's Markov blanket (see page 517).

To analyze the general Gibbs sampler, which samples each X_i in turn with a transition probability q_i that conditions on all the other variables, we define $\overline{\mathbf{X}_i}$ to be these other variables (except the evidence variables); their values in the current state are $\overline{\mathbf{x}_i}$. If we sample a new value x_i' for X_i conditionally on all the other variables, including the evidence, we have

$$q_i(\mathbf{x} \rightarrow \mathbf{x}') = q_i((x_i, \overline{\mathbf{x}_i}) \rightarrow (x_i', \overline{\mathbf{x}_i})) = P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) \ .$$

Now we show that the transition probability for each step of the Gibbs sampler is in detailed balance with the true posterior:

$$\begin{aligned}
&\pi(\mathbf{x}) q_i(\mathbf{x} \rightarrow \mathbf{x}') = P(\mathbf{x} \,|\, \mathbf{e}) P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) = P(x_i, \overline{\mathbf{x}_i} \,|\, \mathbf{e}) P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) \\
&= P(x_i \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) P(\overline{\mathbf{x}_i} \,|\, \mathbf{e}) P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) \qquad \text{(using the chain rule on the first term)} \\
&= P(x_i \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) P(x_i', \overline{\mathbf{x}_i} \,|\, \mathbf{e}) \qquad \text{(using the chain rule backward)} \\
&= \pi(\mathbf{x}') q_i(\mathbf{x}' \rightarrow \mathbf{x}) \ .
\end{aligned}$$

We can think of the loop "**for each** Z_i in **Z do**" in Figure 14.16 as defining one large transition probability q that is the sequential composition $q_1 \circ q_2 \circ \cdots \circ q_n$ of the transition probabilities for the individual variables. It is easy to show (Exercise 14.19) that if each of q_i and q_j has π as its stationary distribution, then the sequential composition $q_i \circ q_j$ does too; hence the transition probability q for the whole loop has $P(\mathbf{x} \,|\, \mathbf{e})$ as its stationary distribution. Finally, unless the CPTs contain probabilities of 0 or 1—which can cause the state space to become disconnected—it is easy to see that q is ergodic. Hence, the samples generated by Gibbs sampling will eventually be drawn from the true posterior distribution.

The final step is to show how to perform the general Gibbs sampling step—sampling X_i from $\mathbf{P}(X_i \,|\, \overline{\mathbf{x}_i}, \mathbf{e})$—in a Bayesian network. Recall from page 517 that a variable is independent of all other variables given its Markov blanket; hence,

$$P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) = P(x_i' \,|\, mb(X_i)) \ ,$$

where $mb(X_i)$ denotes the values of the variables in X_i's Markov blanket, $MB(X_i)$. As shown in Exercise 14.7, the probability of a variable given its Markov blanket is proportional to the probability of the variable given its parents times the probability of each child given its respective parents:

$$P(x_i' \,|\, mb(X_i)) = \alpha \, P(x_i' \,|\, parents(X_i)) \times \prod_{Y_j \in Children(X_i)} P(y_j \,|\, parents(Y_j)) \ . \quad (14.12)$$

Hence, to flip each variable X_i conditioned on its Markov blanket, the number of multiplications required is equal to the number of X_i's children.

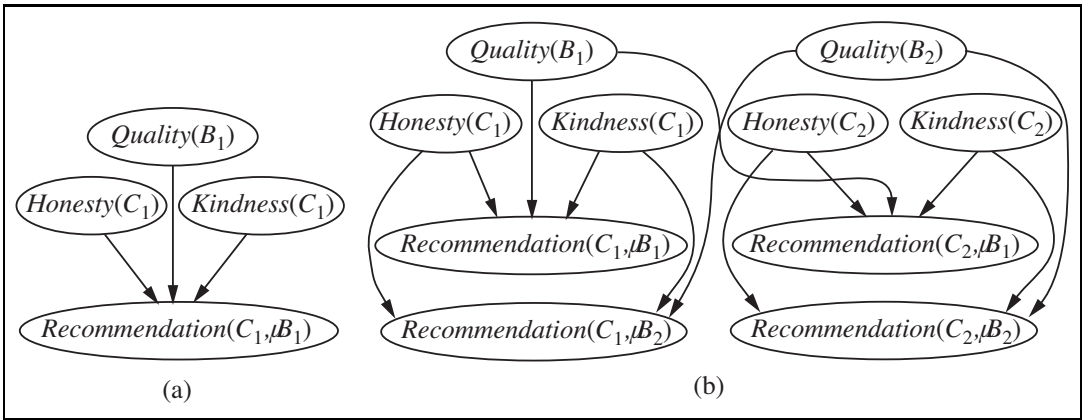


Figure 14.17 (a) Bayes net for a single customer C_1 recommending a single book B_1. $Honest(C_1)$ is Boolean, while the other variables have integer values from 1 to 5. (b) Bayes net with two customers and two books.

14.6 Relational and First-Order Probability Models

In Chapter 8, we explained the representational advantages possessed by first-order logic in comparison to propositional logic. First-order logic commits to the existence of objects and relations among them and can express facts about *some* or *all* of the objects in a domain. This often results in representations that are vastly more concise than the equivalent propositional descriptions. Now, Bayesian networks are essentially propositional: the set of random variables is fixed and finite, and each has a fixed domain of possible values. This fact limits the applicability of Bayesian networks. *If we can find a way to combine probability theory with the expressive power of first-order representations, we expect to be able to increase dramatically the range of problems that can be handled.*

For example, suppose that an online book retailer would like to provide overall evaluations of products based on recommendations received from its customers. The evaluation will take the form of a posterior distribution over the quality of the book, given the available evidence. The simplest solution to base the evaluation on the average recommendation, perhaps with a variance determined by the number of recommendations, but this fails to take into account the fact that some customers are kinder than others and some are less honest than others. Kind customers tend to give high recommendations even to fairly mediocre books, while dishonest customers give very high or very low recommendations for reasons other than quality—for example, they might work for a publisher.[6]

For a single customer C_1, recommending a single book B_1, the Bayes net might look like the one shown in Figure 14.17(a). (Just as in Section 9.1, expressions with parentheses such as $Honest(C_1)$ are just fancy symbols—in this case, fancy names for random variables.)

[6] A game theorist would advise a dishonest customer to avoid detection by occasionally recommending a good book from a competitor. See Chapter 17.

With two customers and two books, the Bayes net looks like the one in Figure 14.17(b). For larger numbers of books and customers, it becomes completely impractical to specify the network by hand.

Fortunately, the network has a lot of repeated structure. Each $Recommendation(c, b)$ variable has as its parents the variables $Honest(c)$, $Kindness(c)$, and $Quality(b)$. Moreover, the CPTs for all the $Recommendation(c, b)$ variables are identical, as are those for all the $Honest(c)$ variables, and so on. The situation seems tailor-made for a first-order language. We would like to say something like

$$Recommendation(c, b) \sim RecCPT(Honest(c), Kindness(c), Quality(b))$$

with the intended meaning that a customer's recommendation for a book depends on the customer's honesty and kindness and the book's quality according to some fixed CPT. This section develops a language that lets us say exactly this, and a lot more besides.

14.6.1 Possible worlds

Recall from Chapter 13 that a probability model defines a set Ω of possible worlds with a probability $P(\omega)$ for each world ω. For Bayesian networks, the possible worlds are assignments of values to variables; for the Boolean case in particular, the possible worlds are identical to those of propositional logic. For a first-order probability model, then, it seems we need the possible worlds to be those of first-order logic—that is, a set of objects with relations among them and an interpretation that maps constant symbols to objects, predicate symbols to relations, and function symbols to functions on those objects. (See Section 8.2.) The model also needs to define a probability for each such possible world, just as a Bayesian network defines a probability for each assignment of values to variables.

Let us suppose, for a moment, that we have figured out how to do this. Then, as usual (see page 485), we can obtain the probability of any first-order logical sentence ϕ as a sum over the possible worlds where it is true:

$$P(\phi) = \sum_{\omega : \phi \text{ is true in } \omega} P(\omega) \ . \tag{14.13}$$

Conditional probabilities $P(\phi \mid \mathbf{e})$ can be obtained similarly, so we can, in principle, ask any question we want of our model—e.g., "Which books are most likely to be recommended highly by dishonest customers?"—and get an answer. So far, so good.

There is, however, a problem: the set of first-order models is infinite. We saw this explicitly in Figure 8.4 on page 293, which we show again in Figure 14.18 (top). This means that (1) the summation in Equation (14.13) could be infeasible, and (2) specifying a complete, consistent distribution over an infinite set of worlds could be very difficult.

Section 14.6.2 explores one approach to dealing with this problem. The idea is to borrow not from the standard semantics of first-order logic but from the **database semantics** defined in Section 8.2.8 (page 299). The database semantics makes the **unique names assumption**—here, we adopt it for the constant symbols. It also assumes **domain closure**—there are no more objects than those that are named. We can then guarantee a finite set of possible worlds by making the set of objects in each world be exactly the set of constant

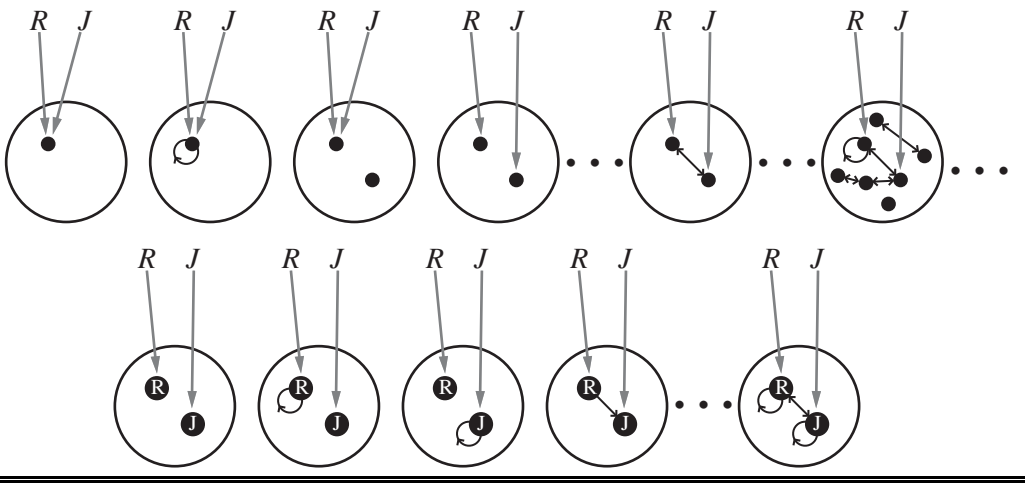


Figure 14.18 Top: Some members of the set of all possible worlds for a language with two constant symbols, R and J, and one binary relation symbol, under the standard semantics for first-order logic. Bottom: the possible worlds under database semantics. The interpretation of the constant symbols is fixed, and there is a distinct object for each constant symbol.

symbols that are used; as shown in Figure 14.18 (bottom), there is no uncertainty about the mapping from symbols to objects or about the objects that exist. We will call models defined in this way **relational probability models**, or RPMs.[7] The most significant difference between the semantics of RPMs and the database semantics introduced in Section 8.2.8 is that RPMs do not make the closed-world assumption—obviously, assuming that every unknown fact is false doesn't make sense in a probabilistic reasoning system!

RELATIONAL PROBABILITY MODEL

When the underlying assumptions of database semantics fail to hold, RPMs won't work well. For example, a book retailer might use an ISBN (International Standard Book Number) as a constant symbol to name each book, even though a given "logical" book (e.g., "Gone With the Wind") may have several ISBNs. It would make sense to aggregate recommendations across multiple ISBNs, but the retailer may not know for sure which ISBNs are really the same book. (Note that we are not reifying the *individual copies* of the book, which might be necessary for used-book sales, car sales, and so on.) Worse still, each customer is identified by a login ID, but a dishonest customer may have thousands of IDs! In the computer security field, these multiple IDs are called **sibyls** and their use to confound a reputation system is called a **sibyl attack**. Thus, even a simple application in a relatively well-defined, online domain involves both **existence uncertainty** (what are the real books and customers underlying the observed data) and **identity uncertainty** (which symbol really refer to the same object). We need to bite the bullet and define probability models based on the standard semantics of first-order logic, for which the possible worlds vary in the objects they contain and in the mappings from symbols to objects. Section 14.6.3 shows how to do this.

SIBYL

SIBYL ATTACK

EXISTENCE UNCERTAINTY

IDENTITY UNCERTAINTY

[7] The name *relational probability model* was given by Pfeffer (2000) to a slightly different representation, but the underlying ideas are the same.

14.6.2 Relational probability models

Like first-order logic, RPMs have constant, function, and predicate symbols. (It turns out to be easier to view predicates as functions that return $true$ or $false$.) We will also assume a

TYPE SIGNATURE

type signature for each function, that is, a specification of the type of each argument and the function's value. If the type of each object is known, many spurious possible worlds are eliminated by this mechanism. For the book-recommendation domain, the types are $Customer$ and $Book$, and the type signatures for the functions and predicates are as follows:

$$Honest : Customer \rightarrow \{true, false\} Kindness : Customer \rightarrow \{1, 2, 3, 4, 5\}$$
$$Quality : Book \rightarrow \{1, 2, 3, 4, 5\}$$
$$Recommendation : Customer \times Book \rightarrow \{1, 2, 3, 4, 5\}$$

The constant symbols will be whatever customer and book names appear in the retailer's data set. In the example given earlier (Figure 14.17(b)), these were $C_1,\ C_2$ and $B_1,\ B_2$.

Given the constants and their types, together with the functions and their type signatures, the random variables of the RPM are obtained by instantiating each function with each possible combination of objects: $Honest(C_1)$, $Quality(B_2)$, $Recommendation(C_1, B_2)$, and so on. These are exactly the variables appearing in Figure 14.17(b). Because each type has only finitely many instances, the number of basic random variables is also finite.

To complete the RPM, we have to write the dependencies that govern these random variables. There is one dependency statement for each function, where each argument of the function is a logical variable (i.e., a variable that ranges over objects, as in first-order logic):

$$Honest(c) \sim \langle 0.99, 0.01 \rangle$$
$$Kindness(c) \sim \langle 0.1, 0.1, 0.2, 0.3, 0.3 \rangle$$
$$Quality(b) \sim \langle 0.05, 0.2, 0.4, 0.2, 0.15 \rangle$$
$$Recommendation(c, b) \sim RecCPT(Honest(c), Kindness(c), Quality(b))$$

where $RecCPT$ is a separately defined conditional distribution with $2 \times 5 \times 5 = 50$ rows, each with 5 entries. The semantics of the RPM can be obtained by instantiating these dependencies for all known constants, giving a Bayesian network (as in Figure 14.17(b)) that defines a joint distribution over the RPM's random variables.[8]

CONTEXT-SPECIFIC INDEPENDENCE

We can refine the model by introducing a **context-specific independence** to reflect the fact that dishonest customers ignore quality when giving a recommendation; moreover, kindness plays no role in their decisions. A context-specific independence allows a variable to be independent of some of its parents given certain values of others; thus, $Recommendation(c, b)$ is independent of $Kindness(c)$ and $Quality(b)$ when $Honest(c) = false$:

$$Recommendation(c, b) \sim \quad \textbf{if } Honest(c) \textbf{ then}$$
$$HonestRecCPT(Kindness(c), Quality(b))$$
$$\textbf{else } \langle 0.4, 0.1, 0.0, 0.1, 0.4 \rangle \ .$$

[8] Some technical conditions must be observed to guarantee that the RPM defines a proper distribution. First, the dependencies must be *acyclic*, otherwise the resulting Bayesian network will have cycles and will not define a proper distribution. Second, the dependencies must be *well-founded*, that is, there can be no infinite ancestor chains, such as might arise from recursive dependencies. Under some circumstances (see Exercise 14.6), a fixed-point calculation yields a well-defined probability model for a recursive RPM.

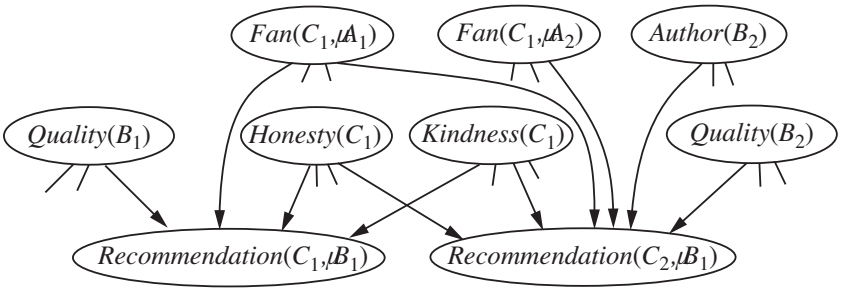


Figure 14.19 Fragment of the equivalent Bayes net when $Author(B_2)$ is unknown.

This kind of dependency may look like an ordinary if–then–else statement on a programming language, but there is a key difference: the inference engine *doesn't necessarily know the value of the conditional test*!

We can elaborate this model in endless ways to make it more realistic. For example, suppose that an honest customer who is a fan of a book's author always gives the book a 5, regardless of quality:

$$\begin{aligned} Recommendation(c, b) \sim \quad & \textbf{if } Honest(c) \textbf{ then} \\ & \quad \textbf{if } Fan(c, Author(b)) \textbf{ then } Exactly(5) \\ & \quad \textbf{else } HonestRecCPT(Kindness(c), Quality(b)) \\ & \textbf{else } \langle 0.4, 0.1, 0.0, 0.1, 0.4 \rangle \end{aligned}$$

Again, the conditional test $Fan(c, Author(b))$ is unknown, but if a customer gives only 5s to a particular author's books and is not otherwise especially kind, then the posterior probability that the customer is a fan of that author will be high. Furthermore, the posterior distribution will tend to discount the customer's 5s in evaluating the quality of that author's books.

In the preceding example, we implicitly assumed that the value of $Author(b)$ is known for every b, but this may not be the case. How can the system reason about whether, say, C_1 is a fan of $Author(B_2)$ when $Author(B_2)$ is unknown? The answer is that the system may have to reason about *all possible authors*. Suppose (to keep things simple) that there are just two authors, A_1 and A_2. Then $Author(B_2)$ is a random variable with two possible values, A_1 and A_2, and it is a parent of $Recommendation(C_1, B_2)$. The variables $Fan(C_1, A_1)$ and $Fan(C_1, A_2)$ are parents too. The conditional distribution for $Recommendation(C_1, B_2)$ is then essentially a **multiplexer** in which the $Author(B_2)$ parent acts as a selector to choose which of $Fan(C_1, A_1)$ and $Fan(C_1, A_2)$ actually gets to influence the recommendation. A fragment of the equivalent Bayes net is shown in Figure 14.19. Uncertainty in the value of $Author(B_2)$, which affects the dependency structure of the network, is an instance of **relational uncertainty**.

MULTIPLEXER

RELATIONAL UNCERTAINTY

In case you are wondering how the system can possibly work out who the author of B_2 is: consider the possibility that three other customers are fans of A_1 (and have no other favorite authors in common) and all three have given B_2 a 5, even though most other customers find it quite dismal. In that case, it is extremely likely that A_1 is the author of B_2.

The emergence of sophisticated reasoning like this from an RPM model of just a few lines is an intriguing example of how probabilistic influences spread through the web of interconnections among objects in the model. As more dependencies and more objects are added, the picture conveyed by the posterior distribution often becomes clearer and clearer.

The next question is how to do inference in RPMs. One approach is to collect the evidence and query and the constant symbols therein, construct the equivalent Bayes net, and apply any of the inference methods discussed in this chapter. This technique is called UNROLLING **unrolling**. The obvious drawback is that the resulting Bayes net may be very large. Furthermore, if there are many candidate objects for an unknown relation or function—for example, the unknown author of B_2—then some variables in the network may have many parents.

Fortunately, much can be done to improve on generic inference algorithms. First, the presence of repeated substructure in the unrolled Bayes net means that many of the factors constructed during variable elimination (and similar kinds of tables constructed by clustering algorithms) will be identical; effective caching schemes have yielded speedups of three orders of magnitude for large networks. Second, inference methods developed to take advantage of context-specific independence in Bayes nets find many applications in RPMs. Third, MCMC inference algorithms have some interesting properties when applied to RPMs with relational uncertainty. MCMC works by sampling complete possible worlds, so in each state the relational structure is completely known. In the example given earlier, each MCMC state would specify the value of $Author(B_2)$, and so the other potential authors are no longer parents of the recommendation nodes for B_2. For MCMC, then, relational uncertainty causes no increase in network complexity; instead, the MCMC process includes transitions that change the relational structure, and hence the dependency structure, of the unrolled network.

All of the methods just described assume that the RPM has to be partially or completely unrolled into a Bayesian network. This is exactly analogous to the method of **propositionalization** for first-order logical inference. (See page 322.) Resolution theorem-provers and logic programming systems avoid propositionalizing by instantiating the logical variables only as needed to make the inference go through; that is, they *lift* the inference process above the level of ground propositional sentences and make each lifted step do the work of many ground steps. The same idea applied in probabilistic inference. For example, in the variable elimination algorithm, a lifted factor can represent an entire set of ground factors that assign probabilities to random variables in the RPM, where those random variables differ only in the constant symbols used to construct them. The details of this method are beyond the scope of this book, but references are given at the end of the chapter.

14.6.3 Open-universe probability models

We argued earlier that database semantics was appropriate for situations in which we know exactly the set of relevant objects that exist and can identify them unambiguously. (In particular, all observations about an object are correctly associated with the constant symbol that names it.) In many real-world settings, however, these assumptions are simply untenable. We gave the examples of multiple ISBNs and sibyl attacks in the book-recommendation domain (to which we will return in a moment), but the phenomenon is far more pervasive:

- A vision system doesn't know what exists, if anything, around the next corner, and may not know if the object it sees now is the same one it saw a few minutes ago.
- A text-understanding system does not know in advance the entities that will be featured in a text, and must reason about whether phrases such as "Mary," "Dr. Smith," "she," "his cardiologist," "his mother," and so on refer to the same object.
- An intelligence analyst hunting for spies never knows how many spies there really are and can only guess whether various pseudonyms, phone numbers, and sightings belong to the same individual.

In fact, a major part of human cognition seems to require learning what objects exist and being able to connect observations—which almost never come with unique IDs attached—to hypothesized objects in the world.

OPEN UNIVERSE

For these reasons, we need to be able to write so-called **open-universe** probability models or OUPMs based on the standard semantics of first-order logic, as illustrated at the top of Figure 14.18. A language for OUPMs provides a way of writing such models easily while guaranteeing a unique, consistent probability distribution over the infinite space of possible worlds.

The basic idea is to understand how ordinary Bayesian networks and RPMs manage to define a unique probability model and to transfer that insight to the first-order setting. In essence, a Bayes net *generates* each possible world, event by event, in the topological order defined by the network structure, where each event is an assignment of a value to a variable. An RPM extends this to entire sets of events, defined by the possible instantiations of the logical variables in a given predicate or function. OUPMs go further by allowing generative steps that *add objects* to the possible world under construction, where the number and type of objects may depend on the objects that are already in that world. That is, the event being generated is not the assignment of a value to a variable, but the very *existence* of objects.

One way to do this in OUPMs is to add statements that define conditional distributions over the numbers of objects of various kinds. For example, in the book-recommendation domain, we might want to distinguish between *customers* (real people) and their *login IDs*. Suppose we expect somewhere between 100 and 10,000 distinct customers (whom we cannot observe directly). We can express this as a prior log-normal distribution[9] as follows:

$$\#\, Customer \sim LogNormal[6.9, 2.3^2]() \; .$$

We expect honest customers to have just one ID, whereas dishonest customers might have anywhere between 10 and 1000 IDs:

$$\#\, LoginID(Owner = c) \sim \quad \textbf{if } Honest(c) \textbf{ then } Exactly(1) \textbf{ else } LogNormal[6.9, 2.3^2]() \; .$$

This statement defines the number of login IDs for a given owner, who is a customer. The

ORIGIN FUNCTION

$Owner$ function is called an **origin function** because it says where each generated object came from. In the formal semantics of BLOG (as distinct from first-order logic), the domain elements in each possible world are actually generation histories (e.g., "the fourth login ID of the seventh customer") rather than simple tokens.

[9] A distribution $LogNormal[\mu, \sigma^2](x)$ is equivalent to a distribution $N[\mu, \sigma^2](x)$ over $\log_e(x)$.

Subject to technical conditions of acyclicity and well-foundedness similar to those for RPMs, open-universe models of this kind define a unique distribution over possible worlds. Furthermore, there exist inference algorithms such that, for every such well-defined model and every first-order query, the answer returned approaches the true posterior arbitrarily closely in the limit. There are some tricky issues involved in designing these algorithms. For example, an MCMC algorithm cannot sample directly in the space of possible worlds when the size of those worlds is unbounded; instead, it samples finite, partial worlds, relying on the fact that only finitely many objects can be relevant to the query in distinct ways. Moreover, transitions must allow for merging two objects into one or splitting one into two. (Details are given in the references at the end of the chapter.) Despite these complications, the basic principle established in Equation (14.13) still holds: the probability of any sentence is well defined and can be calculated.

Research in this area is still at an early stage, but already it is becoming clear that first-order probabilistic reasoning yields a tremendous increase in the effectiveness of AI systems at handling uncertain information. Potential applications include those mentioned above—computer vision, text understanding, and intelligence analysis—as well as many other kinds of sensor interpretation.

14.7 Other Approaches to Uncertain Reasoning

Other sciences (e.g., physics, genetics, and economics) have long favored probability as a model for uncertainty. In 1819, Pierre Laplace said, "Probability theory is nothing but common sense reduced to calculation." In 1850, James Maxwell said, "The true logic for this world is the calculus of Probabilities, which takes account of the magnitude of the probability which is, or ought to be, in a reasonable man's mind."

Given this long tradition, it is perhaps surprising that AI has considered many alternatives to probability. The earliest expert systems of the 1970s ignored uncertainty and used strict logical reasoning, but it soon became clear that this was impractical for most real-world domains. The next generation of expert systems (especially in medical domains) used probabilistic techniques. Initial results were promising, but they did not scale up because of the exponential number of probabilities required in the full joint distribution. (Efficient Bayesian network algorithms were unknown then.) As a result, probabilistic approaches fell out of favor from roughly 1975 to 1988, and a variety of alternatives to probability were tried for a variety of reasons:

- One common view is that probability theory is essentially numerical, whereas human judgmental reasoning is more "qualitative." Certainly, we are not consciously aware of doing numerical calculations of degrees of belief. (Neither are we aware of doing unification, yet we seem to be capable of some kind of logical reasoning.) It might be that we have some kind of numerical degrees of belief encoded directly in strengths of connections and activations in our neurons. In that case, the difficulty of conscious access to those strengths is not surprising. One should also note that qualitative reason-

ing mechanisms can be built directly on top of probability theory, so the "no numbers" argument against probability has little force. Nonetheless, some qualitative schemes have a good deal of appeal in their own right. One of the best studied is **default reasoning**, which treats conclusions not as "believed to a certain degree," but as "believed until a better reason is found to believe something else." Default reasoning is covered in Chapter 12.

- **Rule-based** approaches to uncertainty have also been tried. Such approaches hope to build on the success of logical rule-based systems, but add a sort of "fudge factor" to each rule to accommodate uncertainty. These methods were developed in the mid-1970s and formed the basis for a large number of expert systems in medicine and other areas.
- One area that we have not addressed so far is the question of **ignorance**, as opposed to uncertainty. Consider the flipping of a coin. If we know that the coin is fair, then a probability of 0.5 for heads is reasonable. If we know that the coin is biased, but we do not know which way, then 0.5 for heads is again reasonable. Obviously, the two cases are different, yet the outcome probability seems not to distinguish them. The **Dempster–Shafer theory** uses **interval-valued** degrees of belief to represent an agent's knowledge of the probability of a proposition.
- Probability makes the same ontological commitment as logic: that propositions are true or false in the world, even if the agent is uncertain as to which is the case. Researchers in **fuzzy logic** have proposed an ontology that allows **vagueness**: that a proposition can be "sort of" true. Vagueness and uncertainty are in fact orthogonal issues.

The next three subsections treat some of these approaches in slightly more depth. We will not provide detailed technical material, but we cite references for further study.

14.7.1 Rule-based methods for uncertain reasoning

Rule-based systems emerged from early work on practical and intuitive systems for logical inference. Logical systems in general, and logical rule-based systems in particular, have three desirable properties:

LOCALITY
- **Locality**: In logical systems, whenever we have a rule of the form $A \Rightarrow B$, we can conclude B, given evidence A, *without worrying about any other rules.* In probabilistic systems, we need to consider *all* the evidence.

DETACHMENT
- **Detachment**: Once a logical proof is found for a proposition B, the proposition can be used regardless of how it was derived. That is, it can be **detached** from its justification. In dealing with probabilities, on the other hand, the source of the evidence for a belief is important for subsequent reasoning.

TRUTH-FUNCTIONALITY
- **Truth-functionality**: In logic, the truth of complex sentences can be computed from the truth of the components. Probability combination does not work this way, except under strong global independence assumptions.

There have been several attempts to devise uncertain reasoning schemes that retain these advantages. The idea is to attach degrees of belief to propositions and rules and to devise purely local schemes for combining and propagating those degrees of belief. The schemes

are also truth-functional; for example, the degree of belief in $A \vee B$ is a function of the belief in A and the belief in B.

The bad news for rule-based systems is that the properties of *locality, detachment, and truth-functionality are simply not appropriate for uncertain reasoning.* Let us look at truth-functionality first. Let H_1 be the event that a fair coin flip comes up heads, let T_1 be the event that the coin comes up tails on that same flip, and let H_2 be the event that the coin comes up heads on a second flip. Clearly, all three events have the same probability, 0.5, and so a truth-functional system must assign the same belief to the disjunction of any two of them. But we can see that the probability of the disjunction depends on the events themselves and not just on their probabilities:

$P(A)$	$P(B)$	$P(A \vee B)$
$P(H_1) = 0.5$	$P(H_1) = 0.5$	$P(H_1 \vee H_1) = 0.50$
	$P(T_1) = 0.5$	$P(H_1 \vee T_1) = 1.00$
	$P(H_2) = 0.5$	$P(H_1 \vee H_2) = 0.75$

It gets worse when we chain evidence together. Truth-functional systems have **rules** of the form $A \mapsto B$ that allow us to compute the belief in B as a function of the belief in the rule and the belief in A. Both forward- and backward-chaining systems can be devised. The belief in the rule is assumed to be constant and is usually specified by the knowledge engineer—for example, as $A \mapsto_{0.9} B$.

Consider the wet-grass situation from Figure 14.12(a) (page 529). If we wanted to be able to do both causal and diagnostic reasoning, we would need the two rules

$$Rain \mapsto WetGrass \qquad \text{and} \qquad WetGrass \mapsto Rain\ .$$

These two rules form a feedback loop: evidence for $Rain$ increases the belief in $WetGrass$, which in turn increases the belief in $Rain$ even more. Clearly, uncertain reasoning systems need to keep track of the paths along which evidence is propagated.

Intercausal reasoning (or explaining away) is also tricky. Consider what happens when we have the two rules

$$Sprinkler \mapsto WetGrass \qquad \text{and} \qquad WetGrass \mapsto Rain\ .$$

Suppose we see that the sprinkler is on. Chaining forward through our rules, this increases the belief that the grass will be wet, which in turn increases the belief that it is raining. But this is ridiculous: the fact that the sprinkler is on explains away the wet grass and should *reduce* the belief in rain. A truth-functional system acts as if it also believes $Sprinkler \mapsto Rain$.

Given these difficulties, how can truth-functional systems be made useful in practice? The answer lies in restricting the task and in carefully engineering the rule base so that undesirable interactions do not occur. The most famous example of a truth-functional system for uncertain reasoning is the **certainty factors** model, which was developed for the MYCIN medical diagnosis program and was widely used in expert systems of the late 1970s and 1980s. Almost all uses of certainty factors involved rule sets that were either purely diagnostic (as in MYCIN) or purely causal. Furthermore, evidence was entered only at the "roots" of the rule set, and most rule sets were singly connected. Heckerman (1986) has shown that,

CERTAINTY FACTOR

under these circumstances, a minor variation on certainty-factor inference was exactly equivalent to Bayesian inference on polytrees. In other circumstances, certainty factors could yield disastrously incorrect degrees of belief through overcounting of evidence. As rule sets became larger, undesirable interactions between rules became more common, and practitioners found that the certainty factors of many other rules had to be "tweaked" when new rules were added. For these reasons, Bayesian networks have largely supplanted rule-based methods for uncertain reasoning.

14.7.2 Representing ignorance: Dempster–Shafer theory

DEMPSTER–SHAFER THEORY

The **Dempster–Shafer theory** is designed to deal with the distinction between **uncertainty** and **ignorance**. Rather than computing the probability of a proposition, it computes the probability that the evidence supports the proposition. This measure of belief is called a

BELIEF FUNCTION

belief function, written $Bel(X)$.

We return to coin flipping for an example of belief functions. Suppose you pick a coin from a magician's pocket. Given that the coin might or might not be fair, what belief should you ascribe to the event that it comes up heads? Dempster–Shafer theory says that because you have no evidence either way, you have to say that the belief $Bel(Heads) = 0$ and also that $Bel(\neg Heads) = 0$. This makes Dempster–Shafer reasoning systems skeptical in a way that has some intuitive appeal. Now suppose you have an expert at your disposal who testifies with 90% certainty that the coin is fair (i.e., he is 90% sure that $P(Heads) = 0.5$). Then Dempster–Shafer theory gives $Bel(Heads) = 0.9 \times 0.5 = 0.45$ and likewise $Bel(\neg Heads) = 0.45$. There is still a 10 percentage point "gap" that is not accounted for by the evidence.

The mathematical underpinnings of Dempster–Shafer theory have a similar flavor to those of probability theory; the main difference is that, instead of assigning probabilities

MASS

to possible worlds, the theory assigns **masses** to *sets* of possible world, that is, to events. The masses still must add to 1 over all possible events. $Bel(A)$ is defined to be the sum of masses for all events that are subsets of (i.e., that entail) A, including A itself. With this definition, $Bel(A)$ and $Bel(\neg A)$ sum to *at most* 1, and the gap—the interval between $Bel(A)$ and $1 - Bel(\neg A)$—is often interpreted as bounding the probability of A.

As with default reasoning, there is a problem in connecting beliefs to actions. Whenever there is a gap in the beliefs, then a decision problem can be defined such that a Dempster–Shafer system is unable to make a decision. In fact, the notion of utility in the Dempster–Shafer model is not yet well understood because the meanings of masses and beliefs themselves have yet to be understood. Pearl (1988) has argued that $Bel(A)$ should be interpreted not as a degree of belief in A but as the probability assigned to all the possible worlds (now interpreted as logical theories) in which A is *provable*. While there are cases in which this quantity might be of interest, it is not the same as the probability that A is true.

A Bayesian analysis of the coin-flipping example would suggest that no new formalism is necessary to handle such cases. The model would have two variables: the $Bias$ of the coin (a number between 0 and 1, where 0 is a coin that always shows tails and 1 a coin that always shows heads) and the outcome of the next $Flip$. The prior probability distribution for $Bias$

would reflect our beliefs based on the source of the coin (the magician's pocket): some small probability that it is fair and some probability that it is heavily biased toward heads or tails. The conditional distribution $\mathbf{P}(Flip \mid Bias)$ simply defines how the bias operates. If $\mathbf{P}(Bias)$ is symmetric about 0.5, then our prior probability for the flip is

$$P(Flip = heads) = \int_0^1 P(Bias = x)P(Flip = heads \mid Bias = x)\, dx = 0.5\ .$$

This is the same prediction as if we believe strongly that the coin is fair, but that does *not* mean that probability theory treats the two situations identically. The difference arises *after* the flips in computing the posterior distribution for $Bias$. If the coin came from a bank, then seeing it come up heads three times running would have almost no effect on our strong prior belief in its fairness; but if the coin comes from the magician's pocket, the same evidence will lead to a stronger posterior belief that the coin is biased toward heads. Thus, a Bayesian approach expresses our "ignorance" in terms of how our beliefs would change in the face of future information gathering.

14.7.3 Representing vagueness: Fuzzy sets and fuzzy logic

FUZZY SET THEORY

Fuzzy set theory is a means of specifying how well an object satisfies a vague description. For example, consider the proposition "Nate is tall." Is this true if Nate is $5'\ 10''$? Most people would hesitate to answer "true" or "false," preferring to say, "sort of." Note that this is not a question of uncertainty about the external world—we are sure of Nate's height. The issue is that the linguistic term "tall" does not refer to a sharp demarcation of objects into two classes—there are *degrees* of tallness. For this reason, *fuzzy set theory is not a method for uncertain reasoning at all.* Rather, fuzzy set theory treats $Tall$ as a fuzzy predicate and says that the truth value of $Tall(Nate)$ is a number between 0 and 1, rather than being just $true$ or $false$. The name "fuzzy set" derives from the interpretation of the predicate as implicitly defining a set of its members—a set that does not have sharp boundaries.

FUZZY LOGIC

Fuzzy logic is a method for reasoning with logical expressions describing membership in fuzzy sets. For example, the complex sentence $Tall(Nate) \wedge Heavy(Nate)$ has a fuzzy truth value that is a function of the truth values of its components. The standard rules for evaluating the fuzzy truth, T, of a complex sentence are

$$T(A \wedge B) = \min(T(A), T(B))$$
$$T(A \vee B) = \max(T(A), T(B))$$
$$T(\neg A) = 1 - T(A)\ .$$

Fuzzy logic is therefore a truth-functional system—a fact that causes serious difficulties. For example, suppose that $T(Tall(Nate)) = 0.6$ and $T(Heavy(Nate)) = 0.4$. Then we have $T(Tall(Nate) \wedge Heavy(Nate)) = 0.4$, which seems reasonable, but we also get the result $T(Tall(Nate) \wedge \neg Tall(Nate)) = 0.4$, which does not. Clearly, the problem arises from the inability of a truth-functional approach to take into account the correlations or anticorrelations among the component propositions.

FUZZY CONTROL

Fuzzy control is a methodology for constructing control systems in which the mapping between real-valued input and output parameters is represented by fuzzy rules. Fuzzy control has been very successful in commercial products such as automatic transmissions, video

cameras, and electric shavers. Critics (see, e.g., Elkan, 1993) argue that these applications are successful because they have small rule bases, no chaining of inferences, and tunable parameters that can be adjusted to improve the system's performance. The fact that they are implemented with fuzzy operators might be incidental to their success; the key is simply to provide a concise and intuitive way to specify a smoothly interpolated, real-valued function.

There have been attempts to provide an explanation of fuzzy logic in terms of probability theory. One idea is to view assertions such as "Nate is Tall" as discrete observations made concerning a continuous hidden variable, Nate's actual $Height$. The probability model specifies $P(\text{Observer says Nate is tall} \mid Height)$, perhaps using a **probit distribution** as described on page 522. A posterior distribution over Nate's height can then be calculated in the usual way, for example, if the model is part of a hybrid Bayesian network. Such an approach is not truth-functional, of course. For example, the conditional distribution

$$P(\text{Observer says Nate is tall and heavy} \mid Height, Weight)$$

allows for interactions between height and weight in the causing of the observation. Thus, someone who is eight feet tall and weighs 190 pounds is very unlikely to be called "tall and heavy," even though "eight feet" counts as "tall" and "190 pounds" counts as "heavy."

RANDOM SET

Fuzzy predicates can also be given a probabilistic interpretation in terms of **random sets**—that is, random variables whose possible values are sets of objects. For example, $Tall$ is a random set whose possible values are sets of people. The probability $P(Tall = S_1)$, where S_1 is some particular set of people, is the probability that exactly that set would be identified as "tall" by an observer. Then the probability that "Nate is tall" is the sum of the probabilities of all the sets of which Nate is a member.

Both the hybrid Bayesian network approach and the random sets approach appear to capture aspects of fuzziness without introducing degrees of truth. Nonetheless, there remain many open issues concerning the proper representation of linguistic observations and continuous quantities—issues that have been neglected by most outside the fuzzy community.

14.8 SUMMARY

This chapter has described **Bayesian networks**, a well-developed representation for uncertain knowledge. Bayesian networks play a role roughly analogous to that of propositional logic for definite knowledge.

- A Bayesian network is a directed acyclic graph whose nodes correspond to random variables; each node has a conditional distribution for the node, given its parents.
- Bayesian networks provide a concise way to represent **conditional independence** relationships in the domain.
- A Bayesian network specifies a full joint distribution; each joint entry is defined as the product of the corresponding entries in the local conditional distributions. A Bayesian network is often exponentially smaller than an explicitly enumerated joint distribution.
- Many conditional distributions can be represented compactly by canonical families of

distributions. **Hybrid Bayesian networks**, which include both discrete and continuous variables, use a variety of canonical distributions.

- Inference in Bayesian networks means computing the probability distribution of a set of query variables, given a set of evidence variables. Exact inference algorithms, such as **variable elimination**, evaluate sums of products of conditional probabilities as efficiently as possible.
- In **polytrees** (singly connected networks), exact inference takes time linear in the size of the network. In the general case, the problem is intractable.
- Stochastic approximation techniques such as **likelihood weighting** and **Markov chain Monte Carlo** can give reasonable estimates of the true posterior probabilities in a network and can cope with much larger networks than can exact algorithms.
- Probability theory can be combined with representational ideas from first-order logic to produce very powerful systems for reasoning under uncertainty. **Relational probability models** (RPMs) include representational restrictions that guarantee a well-defined probability distribution that can be expressed as an equivalent Bayesian network. **Open-universe probability models** handle **existence** and **identity uncertainty**, defining probabilty distributions over the infinite space of first-order possible worlds.
- Various alternative systems for reasoning under uncertainty have been suggested. Generally speaking, **truth-functional** systems are not well suited for such reasoning.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The use of networks to represent probabilistic information began early in the 20th century, with the work of Sewall Wright on the probabilistic analysis of genetic inheritance and animal growth factors (Wright, 1921, 1934). I. J. Good (1961), in collaboration with Alan Turing, developed probabilistic representations and Bayesian inference methods that could be regarded as a forerunner of modern Bayesian networks—although the paper is not often cited in this context.[10] The same paper is the original source for the noisy-OR model.

The **influence diagram** representation for decision problems, which incorporated a DAG representation for random variables, was used in decision analysis in the late 1970s (see Chapter 16), but only enumeration was used for evaluation. Judea Pearl developed the message-passing method for carrying out inference in tree networks (Pearl, 1982a) and polytree networks (Kim and Pearl, 1983) and explained the importance of causal rather than diagnostic probability models, in contrast to the certainty-factor systems then in vogue.

The first expert system using Bayesian networks was CONVINCE (Kim, 1983). Early applications in medicine included the MUNIN system for diagnosing neuromuscular disorders (Andersen *et al.*, 1989) and the PATHFINDER system for pathology (Heckerman, 1991). The CPCS system (Pradhan *et al.*, 1994) is a Bayesian network for internal medicine consisting

[10] I. J. Good was chief statistician for Turing's code-breaking team in World War II. In *2001: A Space Odyssey* (Clarke, 1968a), Good and Minsky are credited with making the breakthrough that led to the development of the HAL 9000 computer.

of 448 nodes, 906 links and 8,254 conditional probability values. (The front cover shows a portion of the network.)

Applications in engineering include the Electric Power Research Institute's work on monitoring power generators (Morjaria *et al.*, 1995), NASA's work on displaying time-critical information at Mission Control in Houston (Horvitz and Barry, 1995), and the general field of **network tomography**, which aims to infer unobserved local properties of nodes and links in the Internet from observations of end-to-end message performance (Castro *et al.*, 2004). Perhaps the most widely used Bayesian network systems have been the diagnosis-and-repair modules (e.g., the Printer Wizard) in Microsoft Windows (Breese and Heckerman, 1996) and the Office Assistant in Microsoft Office (Horvitz *et al.*, 1998). Another important application area is biology: Bayesian networks have been used for identifying human genes by reference to mouse genes (Zhang *et al.*, 2003), inferring cellular networks Friedman (2004), and many other tasks in bioinformatics. We could go on, but instead we'll refer you to Pourret *et al.* (2008), a 400-page guide to applications of Bayesian networks.

Ross Shachter (1986), working in the influence diagram community, developed the first complete algorithm for general Bayesian networks. His method was based on goal-directed reduction of the network using posterior-preserving transformations. Pearl (1986) developed a clustering algorithm for exact inference in general Bayesian networks, utilizing a conversion to a directed polytree of clusters in which message passing was used to achieve consistency over variables shared between clusters. A similar approach, developed by the statisticians David Spiegelhalter and Steffen Lauritzen (Lauritzen and Spiegelhalter, 1988), is based on conversion to an undirected form of graphical model called a **Markov network**. This approach is implemented in the HUGIN system, an efficient and widely used tool for uncertain reasoning (Andersen *et al.*, 1989). Boutilier *et al.* (1996) show how to exploit context-specific independence in clustering algorithms.

MARKOV NETWORK

The basic idea of variable elimination—that repeated computations within the overall sum-of-products expression can be avoided by caching—appeared in the symbolic probabilistic inference (SPI) algorithm (Shachter *et al.*, 1990). The elimination algorithm we describe is closest to that developed by Zhang and Poole (1994). Criteria for pruning irrelevant variables were developed by Geiger *et al.* (1990) and by Lauritzen *et al.* (1990); the criterion we give is a simple special case of these. Dechter (1999) shows how the variable elimination idea is essentially identical to **nonserial dynamic programming** (Bertele and Brioschi, 1972), an algorithmic approach that can be applied to solve a range of inference problems in Bayesian networks—for example, finding the **most likely explanation** for a set of observations. This connects Bayesian network algorithms to related methods for solving CSPs and gives a direct measure of the complexity of exact inference in terms of the tree width of the network. Wexler and Meek (2009) describe a method of preventing exponential growth in the size of factors computed in variable elimination; their algorithm breaks down large factors into products of smaller factors and simultaneously computes an error bound for the resulting approximation.

NONSERIAL DYNAMIC PROGRAMMING

The inclusion of continuous random variables in Bayesian networks was considered by Pearl (1988) and Shachter and Kenley (1989); these papers discussed networks containing only continuous variables with linear Gaussian distributions. The inclusion of discrete variables has been investigated by Lauritzen and Wermuth (1989) and implemented in the

cHUGIN system (Olesen, 1993). Further analysis of linear Gaussian models, with connections to many other models used in statistics, appears in Roweis and Ghahramani (1999) The probit distribution is usually attributed to Gaddum (1933) and Bliss (1934), although it had been discovered several times in the 19th century. Bliss's work was expanded considerably by Finney (1947). The probit has been used widely for modeling discrete choice phenomena and can be extended to handle more than two choices (Daganzo, 1979). The logit model was introduced by Berkson (1944); initially much derided, it eventually became more popular than the probit model. Bishop (1995) gives a simple justification for its use.

Cooper (1990) showed that the general problem of inference in unconstrained Bayesian networks is NP-hard, and Paul Dagum and Mike Luby (1993) showed the corresponding approximation problem to be NP-hard. Space complexity is also a serious problem in both clustering and variable elimination methods. The method of **cutset conditioning**, which was developed for CSPs in Chapter 6, avoids the construction of exponentially large tables. In a Bayesian network, a cutset is a set of nodes that, when instantiated, reduces the remaining nodes to a polytree that can be solved in linear time and space. The query is answered by summing over all the instantiations of the cutset, so the overall space requirement is still linear (Pearl, 1988). Darwiche (2001) describes a recursive conditioning algorithm that allows a complete range of space/time tradeoffs.

The development of fast approximation algorithms for Bayesian network inference is a very active area, with contributions from statistics, computer science, and physics. The rejection sampling method is a general technique that is long known to statisticians; it was first applied to Bayesian networks by Max Henrion (1988), who called it **logic sampling**. Likelihood weighting, which was developed by Fung and Chang (1989) and Shachter and Peot (1989), is an example of the well-known statistical method of **importance sampling**. Cheng and Druzdzel (2000) describe an adaptive version of likelihood weighting that works well even when the evidence has very low prior likelihood.

Markov chain Monte Carlo (MCMC) algorithms began with the Metropolis algorithm, due to Metropolis *et al.* (1953), which was also the source of the simulated annealing algorithm described in Chapter 4. The Gibbs sampler was devised by Geman and Geman (1984) for inference in undirected Markov networks. The application of MCMC to Bayesian networks is due to Pearl (1987). The papers collected by Gilks *et al.* (1996) cover a wide variety of applications of MCMC, several of which were developed in the well-known BUGS package (Gilks *et al.*, 1994).

VARIATIONAL APPROXIMATION

There are two very important families of approximation methods that we did not cover in the chapter. The first is the family of **variational approximation** methods, which can be used to simplify complex calculations of all kinds. The basic idea is to propose a reduced version of the original problem that is simple to work with, but that resembles the original problem as closely as possible. The reduced problem is described by some **variational parameters** $\boldsymbol{\lambda}$ that are adjusted to minimize a distance function D between the original and the reduced problem, often by solving the system of equations $\partial D/\partial\boldsymbol{\lambda} = 0$. In many cases, strict upper and lower bounds can be obtained. Variational methods have long been used in statistics (Rustagi, 1976). In statistical physics, the **mean-field** method is a particular variational approximation in which the individual variables making up the model are assumed

VARIATIONAL PARAMETER

MEAN FIELD

to be completely independent. This idea was applied to solve large undirected Markov networks (Peterson and Anderson, 1987; Parisi, 1988). Saul *et al.* (1996) developed the mathematical foundations for applying variational methods to Bayesian networks and obtained accurate lower-bound approximations for sigmoid networks with the use of mean-field methods. Jaakkola and Jordan (1996) extended the methodology to obtain both lower and upper bounds. Since these early papers, variational methods have been applied to many specific families of models. The remarkable paper by Wainwright and Jordan (2008) provides a unifying theoretical analysis of the literature on variational methods.

A second important family of approximation algorithms is based on Pearl's polytree message-passing algorithm (1982a). This algorithm can be applied to general networks, as suggested by Pearl (1988). The results might be incorrect, or the algorithm might fail to terminate, but in many cases, the values obtained are close to the true values. Little attention was paid to this so-called **belief propagation** (or BP) approach until McEliece *et al.* (1998) observed that message passing in a multiply connected Bayesian network was exactly the computation performed by the **turbo decoding** algorithm (Berrou *et al.*, 1993), which provided a major breakthrough in the design of efficient error-correcting codes. The implication is that BP is both fast and accurate on the very large and very highly connected networks used for decoding and might therefore be useful more generally. Murphy *et al.* (1999) presented a promising empirical study of BP's performance, and Weiss and Freeman (2001) established strong convergence results for BP on linear Gaussian networks. Weiss (2000b) shows how an approximation called loopy belief propagation works, and when the approximation is correct. Yedidia *et al.* (2005) made further connections between loopy propagation and ideas from statistical physics.

BELIEF PROPAGATION

TURBO DECODING

The connection between probability and first-order languages was first studied by Carnap (1950). Gaifman (1964) and Scott and Krauss (1966) defined a language in which probabilities could be associated with first-order sentences and for which models were probability measures on possible worlds. Within AI, this idea was developed for propositional logic by Nilsson (1986) and for first-order logic by Halpern (1990). The first extensive investigation of knowledge representation issues in such languages was carried out by Bacchus (1990). The basic idea is that each sentence in the knowledge base expressed a *constraint* on the distribution over possible worlds; one sentence entails another if it expresses a stronger constraint. For example, the sentence $\forall x \;\; P(Hungry(x)) > 0.2$ rules out distributions in which any object is hungry with probability less than 0.2; thus, it entails the sentence $\forall x \;\; P(Hungry(x)) > 0.1$. It turns out that writing a *consistent* set of sentences in these languages is quite difficult and constructing a unique probability model nearly impossible unless one adopts the representation approach of Bayesian networks by writing suitable sentences about conditional probabilities.

Beginning in the early 1990s, researchers working on complex applications noticed the expressive limitations of Bayesian networks and developed various languages for writing "templates" with logical variables, from which large networks could be constructed automatically for each problem instance (Breese, 1992; Wellman *et al.*, 1992). The most important such language was BUGS (Bayesian inference Using Gibbs Sampling) (Gilks *et al.*, 1994), which combined Bayesian networks with the **indexed random variable** notation common in

INDEXED RANDOM VARIABLE

statistics. (In BUGS, an indexed random variable looks like $X[i]$, where i has a defined integer range.) These languages inherited the key property of Bayesian networks: every well-formed knowledge base defines a unique, consistent probability model. Languages with well-defined semantics based on unique names and domain closure drew on the representational capabilities of logic programming (Poole, 1993; Sato and Kameya, 1997; Kersting *et al.*, 2000) and semantic networks (Koller and Pfeffer, 1998; Pfeffer, 2000). Pfeffer (2007) went on to develop IBAL, which represents first-order probability models as probabilistic programs in a programming language extended with a randomization primitive. Another important thread was the combination of relational and first-order notations with (undirected) Markov networks (Taskar *et al.*, 2002; Domingos and Richardson, 2004), where the emphasis has been less on knowledge representation and more on learning from large data sets.

Initially, inference in these models was performed by generating an equivalent Bayesian network. Pfeffer *et al.* (1999) introduced a variable elimination algorithm that cached each computed factor for reuse by later computations involving the same relations but different objects, thereby realizing some of the computational gains of lifting. The first truly lifted inference algorithm was a lifted form of variable elimination described by Poole (2003) and subsequently improved by de Salvo Braz *et al.* (2007). Further advances, including cases where certain aggregate probabilities can be computed in closed form, are described by Milch *et al.* (2008) and Kisynski and Poole (2009). Pasula and Russell (2001) studied the application of MCMC to avoid building the complete equivalent Bayes net in cases of relational and identity uncertainty. Getoor and Taskar (2007) collect many important papers on first-order probability models and their use in machine learning.

RECORD LINKAGE

Probabilistic reasoning about identity uncertainty has two distinct origins. In statistics, the problem of **record linkage** arises when data records do not contain standard unique identifiers—for example, various citations of this book might name its first author "Stuart Russell" or "S. J. Russell" or even "Stewart Russle," and other authors may use the some of the same names. Literally hundreds of companies exist solely to solve record linkage problems in financial, medical, census, and other data. Probabilistic analysis goes back to work by Dunn (1946); the Fellegi–Sunter model (1969), which is essentially naive Bayes applied to matching, still dominates current practice. The second origin for work on identity uncertainty is multitarget tracking (Sittler, 1964), which we cover in Chapter 15. For most of its history, work in symbolic AI assumed erroneously that sensors could supply sentences with unique identifiers for objects. The issue was studied in the context of language understanding by Charniak and Goldman (1992) and in the context of surveillance by (Huang and Russell, 1998) and Pasula *et al.* (1999). Pasula *et al.* (2003) developed a complex generative model for authors, papers, and citation strings, involving both relational and identity uncertainty, and demonstrated high accuracy for citation information extraction. The first formally defined language for open-universe probability models was BLOG (Milch *et al.*, 2005), which came with a complete (albeit slow) MCMC inference algorithm for all well-defined mdoels. (The program code faintly visible on the front cover of this book is part of a BLOG model for detecting nuclear explosions from seismic signals as part of the UN Comprehensive Test Ban Treaty verification regime.) Laskey (2008) describes another open-universe modeling language called **multi-entity Bayesian networks**.

As explained in Chapter 13, early probabilistic systems fell out of favor in the early 1970s, leaving a partial vacuum to be filled by alternative methods. Certainty factors were invented for use in the medical expert system MYCIN (Shortliffe, 1976), which was intended both as an engineering solution and as a model of human judgment under uncertainty. The collection *Rule-Based Expert Systems* (Buchanan and Shortliffe, 1984) provides a complete overview of MYCIN and its descendants (see also Stefik, 1995). David Heckerman (1986) showed that a slightly modified version of certainty factor calculations gives correct probabilistic results in some cases, but results in serious overcounting of evidence in other cases. The PROSPECTOR expert system (Duda *et al.*, 1979) used a rule-based approach in which the rules were justified by a (seldom tenable) global independence assumption.

Dempster–Shafer theory originates with a paper by Arthur Dempster (1968) proposing a generalization of probability to interval values and a combination rule for using them. Later work by Glenn Shafer (1976) led to the Dempster-Shafer theory's being viewed as a competing approach to probability. Pearl (1988) and Ruspini *et al.* (1992) analyze the relationship between the Dempster–Shafer theory and standard probability theory.

Fuzzy sets were developed by Lotfi Zadeh (1965) in response to the perceived difficulty of providing exact inputs to intelligent systems. The text by Zimmermann (2001) provides a thorough introduction to fuzzy set theory; papers on fuzzy applications are collected in Zimmermann (1999). As we mentioned in the text, fuzzy logic has often been perceived incorrectly as a direct competitor to probability theory, whereas in fact it addresses a different set of issues. **Possibility theory** (Zadeh, 1978) was introduced to handle uncertainty in fuzzy systems and has much in common with probability. Dubois and Prade (1994) survey the connections between possibility theory and probability theory.

POSSIBILITY THEORY

The resurgence of probability depended mainly on Pearl's development of Bayesian networks as a method for representing and using conditional independence information. This resurgence did not come without a fight; Peter Cheeseman's (1985) pugnacious "In Defense of Probability" and his later article "An Inquiry into Computer Understanding" (Cheeseman, 1988, with commentaries) give something of the flavor of the debate. Eugene Charniak helped present the ideas to AI researchers with a popular article, "Bayesian networks without tears"[11] (1991), and book (1993). The book by Dean and Wellman (1991) also helped introduce Bayesian networks to AI researchers. One of the principal philosophical objections of the logicists was that the numerical calculations that probability theory was thought to require were not apparent to introspection and presumed an unrealistic level of precision in our uncertain knowledge. The development of **qualitative probabilistic networks** (Wellman, 1990a) provided a purely qualitative abstraction of Bayesian networks, using the notion of positive and negative influences between variables. Wellman shows that in many cases such information is sufficient for optimal decision making without the need for the precise specification of probability values. Goldszmidt and Pearl (1996) take a similar approach. Work by Adnan Darwiche and Matt Ginsberg (1992) extracts the basic properties of conditioning and evidence combination from probability theory and shows that they can also be applied in logical and default reasoning. Often, programs speak louder than words, and the ready avail-

[11] The title of the original version of the article was "Pearl for swine."

ability of high-quality software such as the Bayes Net toolkit (Murphy, 2001) accelerated the adoption of the technology.

The most important single publication in the growth of Bayesian networks was undoubtedly the text *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). Several excellent texts (Lauritzen, 1996; Jensen, 2001; Korb and Nicholson, 2003; Jensen, 2007; Darwiche, 2009; Koller and Friedman, 2009) provide thorough treatments of the topics we have covered in this chapter. New research on probabilistic reasoning appears both in mainstream AI journals, such as *Artificial Intelligence* and the *Journal of AI Research*, and in more specialized journals, such as the *International Journal of Approximate Reasoning*. Many papers on graphical models, which include Bayesian networks, appear in statistical journals. The proceedings of the conferences on Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NIPS), and Artificial Intelligence and Statistics (AISTATS) are excellent sources for current research.

EXERCISES

14.1 We have a bag of three biased coins a, b, and c with probabilities of coming up heads of 20%, 60%, and 80%, respectively. One coin is drawn randomly from the bag (with equal likelihood of drawing each of the three coins), and then the coin is flipped three times to generate the outcomes X_1, X_2, and X_3.

a. Draw the Bayesian network corresponding to this setup and define the necessary CPTs.

b. Calculate which coin was most likely to have been drawn from the bag if the observed flips come out heads twice and tails once.

14.2 Equation (14.1) on page 513 defines the joint distribution represented by a Bayesian network in terms of the parameters $\theta(X_i \mid Parents(X_i))$. This exercise asks you to derive the equivalence between the parameters and the conditional probabilities $\mathbf{P}(X_i \mid Parents(X_i))$ from this definition.

a. Consider a simple network $X \rightarrow Y \rightarrow Z$ with three Boolean variables. Use Equations (13.3) and (13.6) (pages 485 and 492) to express the conditional probability $P(z \mid y)$ as the ratio of two sums, each over entries in the joint distribution $\mathbf{P}(X, Y, Z)$.

b. Now use Equation (14.1) to write this expression in terms of the network parameters $\theta(X)$, $\theta(Y \mid X)$, and $\theta(Z \mid Y)$.

c. Next, expand out the summations in your expression from part (b), writing out explicitly the terms for the true and false values of each summed variable. Assuming that all network parameters satisfy the constraint $\sum_{x_i} \theta(x_i \mid parents(X_i)) = 1$, show that the resulting expression reduces to $\theta(x \mid y)$.

d. Generalize this derivation to show that $\theta(X_i \mid Parents(X_i)) = \mathbf{P}(X_i \mid Parents(X_i))$ for any Bayesian network.

ARC REVERSAL

14.3 The operation of **arc reversal** in a Bayesian network allows us to change the direction of an arc $X \rightarrow Y$ while preserving the joint probability distribution that the network represents (Shachter, 1986). Arc reversal may require introducing new arcs: all the parents of X also become parents of Y, and all parents of Y also become parents of X.

a. Assume that X and Y start with m and n parents, respectively, and that all variables have k values. By calculating the change in size for the CPTs of X and Y, show that the total number of parameters in the network cannot decrease during arc reversal. (*Hint*: the parents of X and Y need not be disjoint.)

b. Under what circumstances can the total number remain constant?

c. Let the parents of X be $\mathbf{U} \cup \mathbf{V}$ and the parents of Y be $\mathbf{V} \cup \mathbf{W}$, where $\mathbf{U}$ and $\mathbf{W}$ are disjoint. The formulas for the new CPTs after arc reversal are as follows:

$$\begin{aligned}\mathbf{P}(Y \mid \mathbf{U}, \mathbf{V}, \mathbf{W}) &= \sum_x \mathbf{P}(Y \mid \mathbf{V}, \mathbf{W}, x)\mathbf{P}(x \mid \mathbf{U}, \mathbf{V}) \\ \mathbf{P}(X \mid \mathbf{U}, \mathbf{V}, \mathbf{W}, Y) &= \mathbf{P}(Y \mid X, \mathbf{V}, \mathbf{W})\mathbf{P}(X \mid \mathbf{U}, \mathbf{V}) / \mathbf{P}(Y \mid \mathbf{U}, \mathbf{V}, \mathbf{W}) \ .\end{aligned}$$

Prove that the new network expresses the same joint distribution over all variables as the original network.

14.4 Consider the Bayesian network in Figure 14.2.

a. If no evidence is observed, are $Burglary$ and $Earthquake$ independent? Prove this from the numerical semantics and from the topological semantics.

b. If we observe $Alarm = true$, are $Burglary$ and $Earthquake$ independent? Justify your answer by calculating whether the probabilities involved satisfy the definition of conditional independence.

14.5 Suppose that in a Bayesian network containing an unobserved variable Y, all the variables in the Markov blanket $MB(Y)$ have been observed.

a. Prove that removing the node Y from the network will not affect the posterior distribution for any other unobserved variable in the network.

b. Discuss whether we can remove Y if we are planning to use (i) rejection sampling and (ii) likelihood weighting.

14.6 Let H_x be a random variable denoting the handedness of an individual x, with possible values l or r. A common hypothesis is that left- or right-handedness is inherited by a simple mechanism; that is, perhaps there is a gene G_x, also with values l or r, and perhaps actual handedness turns out mostly the same (with some probability s) as the gene an individual possesses. Furthermore, perhaps the gene itself is equally likely to be inherited from either of an individual's parents, with a small nonzero probability m of a random mutation flipping the handedness.

a. Which of the three networks in Figure 14.20 claim that $\mathbf{P}(G_{father}, G_{mother}, G_{child}) = \mathbf{P}(G_{father})\mathbf{P}(G_{mother})\mathbf{P}(G_{child})$?

b. Which of the three networks make independence claims that are consistent with the hypothesis about the inheritance of handedness?

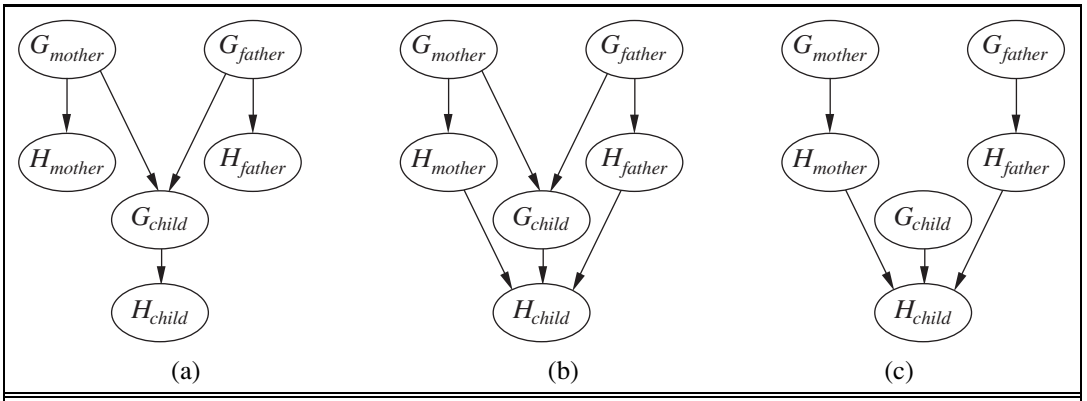


Figure 14.20 Three possible structures for a Bayesian network describing genetic inheritance of handedness.

c. Which of the three networks is the best description of the hypothesis?

d. Write down the CPT for the G_{child} node in network (a), in terms of s and m.

e. Suppose that $P(G_{father} = l) = P(G_{mother} = l) = q$. In network (a), derive an expression for $P(G_{child} = l)$ in terms of m and q only, by conditioning on its parent nodes.

f. Under conditions of genetic equilibrium, we expect the distribution of genes to be the same across generations. Use this to calculate the value of q, and, given what you know about handedness in humans, explain why the hypothesis described at the beginning of this question must be wrong.

14.7 The **Markov blanket** of a variable is defined on page 517. Prove that a variable is independent of all other variables in the network, given its Markov blanket and derive Equation (14.12) (page 538).

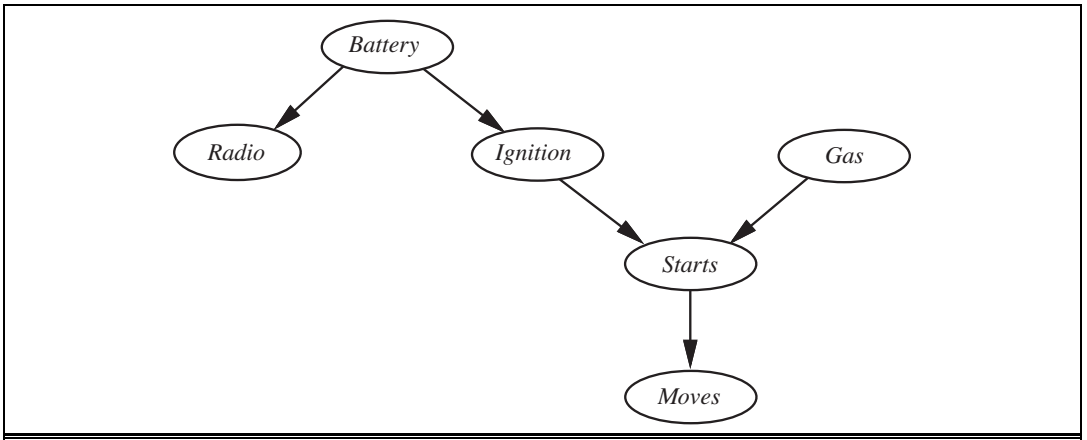


Figure 14.21 A Bayesian network describing some features of a car's electrical system and engine. Each variable is Boolean, and the *true* value indicates that the corresponding aspect of the vehicle is in working order.

14.8 Consider the network for car diagnosis shown in Figure 14.21.

a. Extend the network with the Boolean variables $IcyWeather$ and $StarterMotor$.

b. Give reasonable conditional probability tables for all the nodes.

c. How many independent values are contained in the joint probability distribution for eight Boolean nodes, assuming that no conditional independence relations are known to hold among them?

d. How many independent probability values do your network tables contain?

e. The conditional distribution for $Starts$ could be described as a **noisy-AND** distribution. Define this family in general and relate it to the noisy-OR distribution.

14.9 Consider the family of linear Gaussian networks, as defined on page 520.

a. In a two-variable network, let X_1 be the parent of X_2, let X_1 have a Gaussian prior, and let $\mathbf{P}(X_2 \mid X_1)$ be a linear Gaussian distribution. Show that the joint distribution $P(X_1, X_2)$ is a multivariate Gaussian, and calculate its covariance matrix.

b. Prove by induction that the joint distribution for a general linear Gaussian network on $X_1, \ldots, X_n$ is also a multivariate Gaussian.

14.10 The probit distribution defined on page 522 describes the probability distribution for a Boolean child, given a single continuous parent.

a. How might the definition be extended to cover multiple continuous parents?

b. How might it be extended to handle a *multivalued* child variable? Consider both cases where the child's values are ordered (as in selecting a gear while driving, depending on speed, slope, desired acceleration, etc.) and cases where they are unordered (as in selecting bus, train, or car to get to work). (*Hint*: Consider ways to divide the possible values into two sets, to mimic a Boolean variable.)

14.11 In your local nuclear power station, there is an alarm that senses when a temperature gauge exceeds a given threshold. The gauge measures the temperature of the core. Consider the Boolean variables A (alarm sounds), F_A (alarm is faulty), and F_G (gauge is faulty) and the multivalued nodes G (gauge reading) and T (actual core temperature).

a. Draw a Bayesian network for this domain, given that the gauge is more likely to fail when the core temperature gets too high.

b. Is your network a polytree? Why or why not?

c. Suppose there are just two possible actual and measured temperatures, normal and high; the probability that the gauge gives the correct temperature is x when it is working, but y when it is faulty. Give the conditional probability table associated with G.

d. Suppose the alarm works correctly unless it is faulty, in which case it never sounds. Give the conditional probability table associated with A.

e. Suppose the alarm and gauge are working and the alarm sounds. Calculate an expression for the probability that the temperature of the core is too high, in terms of the various conditional probabilities in the network.

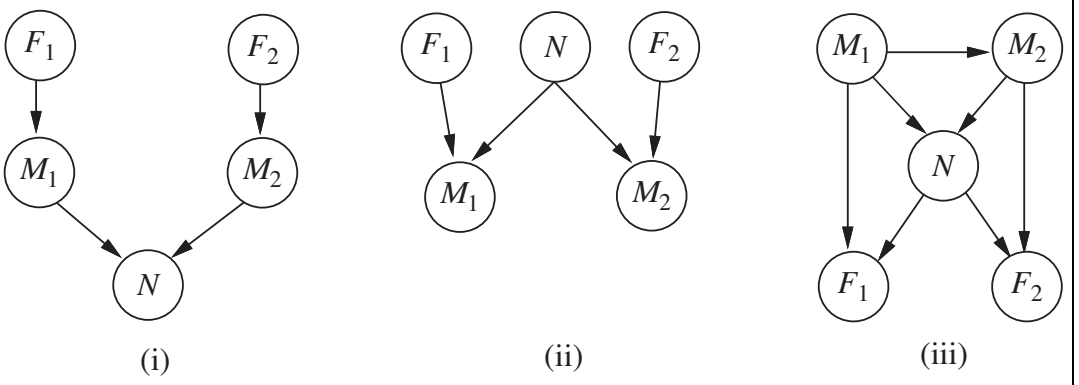


Figure 14.22 Three possible networks for the telescope problem.

14.12 Two astronomers in different parts of the world make measurements M_1 and M_2 of the number of stars N in some small region of the sky, using their telescopes. Normally, there is a small possibility e of error by up to one star in each direction. Each telescope can also (with a much smaller probability f) be badly out of focus (events F_1 and F_2), in which case the scientist will undercount by three or more stars (or if N is less than 3, fail to detect any stars at all). Consider the three networks shown in Figure 14.22.

a. Which of these Bayesian networks are correct (but not necessarily efficient) representations of the preceding information?

b. Which is the best network? Explain.

c. Write out a conditional distribution for $\mathbf{P}(M_1 \mid N)$, for the case where $N \in \{1, 2, 3\}$ and $M_1 \in \{0, 1, 2, 3, 4\}$. Each entry in the conditional distribution should be expressed as a function of the parameters e and/or f.

d. Suppose $M_1 = 1$ and $M_2 = 3$. What are the *possible* numbers of stars if you assume no prior constraint on the values of N?

e. What is the *most likely* number of stars, given these observations? Explain how to compute this, or if it is not possible to compute, explain what additional information is needed and how it would affect the result.

14.13 Consider the network shown in Figure 14.22(ii), and assume that the two telescopes work identically. $N \in \{1, 2, 3\}$ and $M_1, M_2 \in \{0, 1, 2, 3, 4\}$, with the symbolic CPTs as described in Exercise 14.12. Using the enumeration algorithm (Figure 14.9 on page 525), calculate the probability distribution $\mathbf{P}(N \mid M_1 = 2, M_2 = 2)$.

14.14 Consider the Bayes net shown in Figure 14.23.

a. Which of the following are asserted by the network *structure*?

(i) $\mathbf{P}(B, I, M) = \mathbf{P}(B)\mathbf{P}(I)\mathbf{P}(M)$.

(ii) $\mathbf{P}(J \mid G) = \mathbf{P}(J \mid G, I)$.

(iii) $\mathbf{P}(M \mid G, B, I) = \mathbf{P}(M \mid G, B, I, J)$.

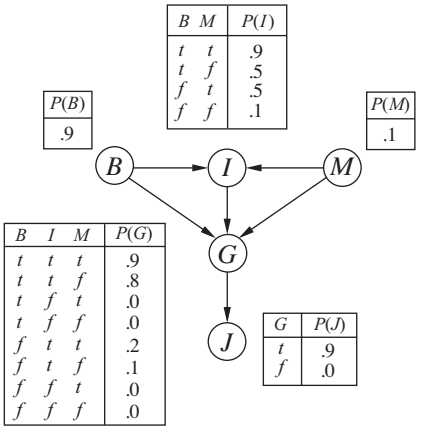


Figure 14.23 A simple Bayes net with Boolean variables $B = BrokeElectionLaw$, $I = Indicted$, $M = PoliticallyMotivatedProsecutor$, $G = FoundGuilty$, $J = Jailed$.

b. Calculate the value of $P(b, i, \neg m, g, j)$.

c. Calculate the probability that someone goes to jail given that they broke the law, have been indicted, and face a politically motivated prosecutor.

d. A **context-specific independence** (see page 542) allows a variable to be independent of some of its parents given certain values of others. In addition to the usual conditional independences given by the graph structure, what context-specific independences exist in the Bayes net in Figure 14.23?

e. Suppose we want to add the variable $P = PresidentialPardon$ to the network; draw the new network and briefly explain any links you add.

14.15 Consider the variable elimination algorithm in Figure 14.11 (page 528).

a. Section 14.4 applies variable elimination to the query

$$\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true) .$$

Perform the calculations indicated and check that the answer is correct.

b. Count the number of arithmetic operations performed, and compare it with the number performed by the enumeration algorithm.

c. Suppose a network has the form of a *chain*: a sequence of Boolean variables $X_1, \ldots, X_n$ where $Parents(X_i) = \{X_{i-1}\}$ for $i = 2, \ldots, n$. What is the complexity of computing $\mathbf{P}(X_1 \mid X_n = true)$ using enumeration? Using variable elimination?

d. Prove that the complexity of running variable elimination on a polytree network is linear in the size of the tree for any variable ordering consistent with the network structure.

14.16 Investigate the complexity of exact inference in general Bayesian networks:

a. Prove that any 3-SAT problem can be reduced to exact inference in a Bayesian network constructed to represent the particular problem and hence that exact inference is NP-

hard. (*Hint*: Consider a network with one variable for each proposition symbol, one for each clause, and one for the conjunction of clauses.)

b. The problem of counting the number of satisfying assignments for a 3-SAT problem is #P-complete. Show that exact inference is at least as hard as this.

14.17 Consider the problem of generating a random sample from a specified distribution on a single variable. Assume you have a random number generator that returns a random number uniformly distributed between 0 and 1.

CUMULATIVE DISTRIBUTION

a. Let X be a discrete variable with $P(X = x_i) = p_i$ for $i \in \{1, \ldots, k\}$. The **cumulative distribution** of X gives the probability that $X \in \{x_1, \ldots, x_j\}$ for each possible j. (See also Appendix A.) Explain how to calculate the cumulative distribution in $O(k)$ time and how to generate a single sample of X from it. Can the latter be done in less than $O(k)$ time?

b. Now suppose we want to generate N samples of X, where $N \gg k$. Explain how to do this with an expected run time per sample that is *constant* (i.e., independent of k).

c. Now consider a continuous-valued variable with a parameterized distribution (e.g., Gaussian). How can samples be generated from such a distribution?

d. Suppose you want to query a continuous-valued variable and you are using a sampling algorithm such as LIKELIHOODWEIGHTING to do the inference. How would you have to modify the query-answering process?

14.18 Consider the query $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true)$ in Figure 14.12(a) (page 529) and how Gibbs sampling can answer it.

a. How many states does the Markov chain have?

b. Calculate the **transition matrix Q** containing $q(\mathbf{y} \rightarrow \mathbf{y}')$ for all $\mathbf{y}$, $\mathbf{y}'$.

c. What does $\mathbf{Q}^2$, the square of the transition matrix, represent?

d. What about $\mathbf{Q}^n$ as $n \rightarrow \infty$?

e. Explain how to do probabilistic inference in Bayesian networks, assuming that $\mathbf{Q}^n$ is available. Is this a practical way to do inference?

14.19 This exercise explores the stationary distribution for Gibbs sampling methods.

a. The convex composition $[\alpha, q_1; 1 - \alpha, q_2]$ of q_1 and q_2 is a transition probability distribution that first chooses one of q_1 and q_2 with probabilities α and $1 - \alpha$, respectively, and then applies whichever is chosen. Prove that if q_1 and q_2 are in detailed balance with π, then their convex composition is also in detailed balance with π. (*Note*: this result justifies a variant of GIBBS-ASK in which variables are chosen at random rather than sampled in a fixed sequence.)

b. Prove that if each of q_1 and q_2 has π as its stationary distribution, then the sequential composition $q = q_1 \circ q_2$ also has π as its stationary distribution.

METROPOLIS–HASTINGS

14.20 The **Metropolis–Hastings** algorithm is a member of the MCMC family; as such, it is designed to generate samples $\mathbf{x}$ (eventually) according to target probabilities $\pi(\mathbf{x})$. (Typically

we are interested in sampling from $\pi(\mathbf{x}) = P(\mathbf{x} \mid \mathbf{e})$.) Like simulated annealing, Metropolis–Hastings operates in two stages. First, it samples a new state $\mathbf{x}'$ from a **proposal distribution** $q(\mathbf{x}' \mid \mathbf{x})$, given the current state $\mathbf{x}$. Then, it probabilistically accepts or rejects $\mathbf{x}'$ according to the **acceptance probability**

PROPOSAL DISTRIBUTION

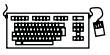
ACCEPTANCE PROBABILITY

$$\alpha(\mathbf{x}' \mid \mathbf{x}) = \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x})}\right) .$$

If the proposal is rejected, the state remains at $\mathbf{x}$.

a. Consider an ordinary Gibbs sampling step for a specific variable X_i. Show that this step, considered as a proposal, is guaranteed to be accepted by Metropolis–Hastings. (Hence, Gibbs sampling is a special case of Metropolis–Hastings.)

b. Show that the two-step process above, viewed as a transition probability distribution, is in detailed balance with π.



14.21 Three soccer teams A, B, and C, play each other once. Each match is between two teams, and can be won, drawn, or lost. Each team has a fixed, unknown degree of quality—an integer ranging from 0 to 3—and the outcome of a match depends probabilistically on the difference in quality between the two teams.

a. Construct a relational probability model to describe this domain, and suggest numerical values for all the necessary probability distributions.

b. Construct the equivalent Bayesian network for the three matches.

c. Suppose that in the first two matches A beats B and draws with C. Using an exact inference algorithm of your choice, compute the posterior distribution for the outcome of the third match.

d. Suppose there are n teams in the league and we have the results for all but the last match. How does the complexity of predicting the last game vary with n?

e. Investigate the application of MCMC to this problem. How quickly does it converge in practice and how well does it scale?

15 PROBABILISTIC REASONING OVER TIME

In which we try to interpret the present, understand the past, and perhaps predict the future, even when very little is crystal clear.

Agents in partially observable environments must be able to keep track of the current state, to the extent that their sensors allow. In Section 4.4 we showed a methodology for doing that: an agent maintains a **belief state** that represents which states of the world are currently possible. From the belief state and a **transition model**, the agent can predict how the world might evolve in the next time step. From the percepts observed and a **sensor model**, the agent can update the belief state. This is a pervasive idea: in Chapter 4 belief states were represented by explicitly enumerated sets of states, whereas in Chapters 7 and 11 they were represented by logical formulas. Those approaches defined belief states in terms of which world states were *possible*, but could say nothing about which states were *likely* or *unlikely*. In this chapter, we use probability theory to quantify the degree of belief in elements of the belief state.

As we show in Section 15.1, time itself is handled in the same way as in Chapter 7: a changing world is modeled using a variable for each aspect of the world state *at each point in time*. The transition and sensor models may be uncertain: the transition model describes the probability distribution of the variables at time t, given the state of the world at past times, while the sensor model describes the probability of each percept at time t, given the current state of the world. Section 15.2 defines the basic inference tasks and describes the general structure of inference algorithms for temporal models. Then we describe three specific kinds of models: **hidden Markov models**, **Kalman filters**, and **dynamic Bayesian networks** (which include hidden Markov models and Kalman filters as special cases). Finally, Section 15.6 examines the problems faced when keeping track of more than one thing.

15.1 TIME AND UNCERTAINTY

We have developed our techniques for probabilistic reasoning in the context of *static* worlds, in which each random variable has a single fixed value. For example, when repairing a car, we assume that whatever is broken remains broken during the process of diagnosis; our job is to infer the state of the car from observed evidence, which also remains fixed.

Now consider a slightly different problem: treating a diabetic patient. As in the case of car repair, we have evidence such as recent insulin doses, food intake, blood sugar measurements, and other physical signs. The task is to assess the current state of the patient, including the actual blood sugar level and insulin level. Given this information, we can make a decision about the patient's food intake and insulin dose. Unlike the case of car repair, here the *dynamic* aspects of the problem are essential. Blood sugar levels and measurements thereof can change rapidly over time, depending on recent food intake and insulin doses, metabolic activity, the time of day, and so on. To assess the current state from the history of evidence and to predict the outcomes of treatment actions, we must model these changes.

The same considerations arise in many other contexts, such as tracking the location of a robot, tracking the economic activity of a nation, and making sense of a spoken or written sequence of words. How can dynamic situations like these be modeled?

15.1.1 States and observations

TIME SLICE

We view the world as a series of snapshots, or **time slices**, each of which contains a set of random variables, some observable and some not.[1] For simplicity, we will assume that the same subset of variables is observable in each time slice (although this is not strictly necessary in anything that follows). We will use $\mathbf{X}_t$ to denote the set of state variables at time t, which are assumed to be unobservable, and $\mathbf{E}_t$ to denote the set of observable evidence variables. The observation at time t is $\mathbf{E}_t = \mathbf{e}_t$ for some set of values $\mathbf{e}_t$.

Consider the following example: You are the security guard stationed at a secret underground installation. You want to know whether it's raining today, but your only access to the outside world occurs each morning when you see the director coming in with, or without, an umbrella. For each day t, the set $\mathbf{E}_t$ thus contains a single evidence variable $Umbrella_t$ or U_t for short (whether the umbrella appears), and the set $\mathbf{X}_t$ contains a single state variable $Rain_t$ or R_t for short (whether it is raining). Other problems can involve larger sets of variables. In the diabetes example, we might have evidence variables, such as $MeasuredBloodSugar_t$ and $PulseRate_t$, and state variables, such as $BloodSugar_t$ and $StomachContents_t$. (Notice that $BloodSugar_t$ and $MeasuredBloodSugar_t$ are not the same variable; this is how we deal with noisy measurements of actual quantities.)

The interval between time slices also depends on the problem. For diabetes monitoring, a suitable interval might be an hour rather than a day. In this chapter we assume the interval between slices is fixed, so we can label times by integers. We will assume that the state sequence starts at $t = 0$; for various uninteresting reasons, we will assume that evidence starts arriving at $t = 1$ rather than $t = 0$. Hence, our umbrella world is represented by state variables R_0, R_1, R_2, ... and evidence variables U_1, U_2, ... We will use the notation $a{:}b$ to denote the sequence of integers from a to b (inclusive), and the notation $\mathbf{X}_{a:b}$ to denote the set of variables from $\mathbf{X}_a$ to $\mathbf{X}_b$. For example, $U_{1:3}$ corresponds to the variables U_1, U_2, U_3.

[1] Uncertainty over *continuous* time can be modeled by **stochastic differential equations** (SDEs). The models studied in this chapter can be viewed as discrete-time approximations to SDEs.

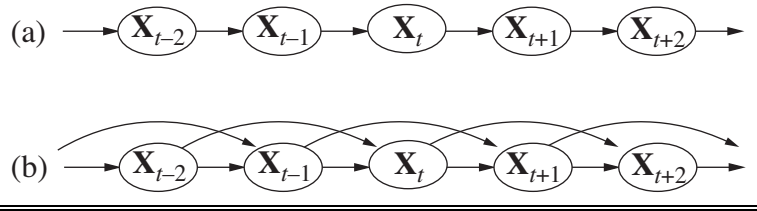


Figure 15.1 (a) Bayesian network structure corresponding to a first-order Markov process with state defined by the variables $\mathbf{X}_t$. (b) A second-order Markov process.

15.1.2 Transition and sensor models

With the set of state and evidence variables for a given problem decided on, the next step is to specify how the world evolves (the transition model) and how the evidence variables get their values (the sensor model).

The transition model specifies the probability distribution over the latest state variables, given the previous values, that is, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{0:t-1})$. Now we face a problem: the set $\mathbf{X}_{0:t-1}$ is unbounded in size as t increases. We solve the problem by making a **Markov assumption**—that the current state depends on only a *finite fixed number* of previous states. Processes satisfying this assumption were first studied in depth by the Russian statistician Andrei Markov (1856–1922) and are called **Markov processes** or **Markov chains**. They come in various flavors; the simplest is the **first-order Markov process**, in which the current state depends only on the previous state and not on any earlier states. In other words, a state provides enough information to make the future conditionally independent of the past, and we have

MARKOV ASSUMPTION

MARKOV PROCESS

FIRST-ORDER MARKOV PROCESS

$$\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1}) \ . \tag{15.1}$$

Hence, in a first-order Markov process, the transition model is the conditional distribution $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$. The transition model for a second-order Markov process is the conditional distribution $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$. Figure 15.1 shows the Bayesian network structures corresponding to first-order and second-order Markov processes.

Even with the Markov assumption there is still a problem: there are infinitely many possible values of t. Do we need to specify a different distribution for each time step? We avoid this problem by assuming that changes in the world state are caused by a **stationary process**—that is, a process of change that is governed by laws that do not themselves change over time. (Don't confuse *stationary* with *static*: in a *static* process, the state itself does not change.) In the umbrella world, then, the conditional probability of rain, $\mathbf{P}(R_t \mid R_{t-1})$, is the same for all t, and we only have to specify one conditional probability table.

STATIONARY PROCESS

Now for the sensor model. The evidence variables $\mathbf{E}_t$ *could* depend on previous variables as well as the current state variables, but any state that's worth its salt should suffice to generate the current sensor values. Thus, we make a **sensor Markov assumption** as follows:

SENSOR MARKOV ASSUMPTION

$$\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t) \ . \tag{15.2}$$

Thus, $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$ is our sensor model (sometimes called the **observation model**). Figure 15.2 shows both the transition model and the sensor model for the umbrella example. Notice the

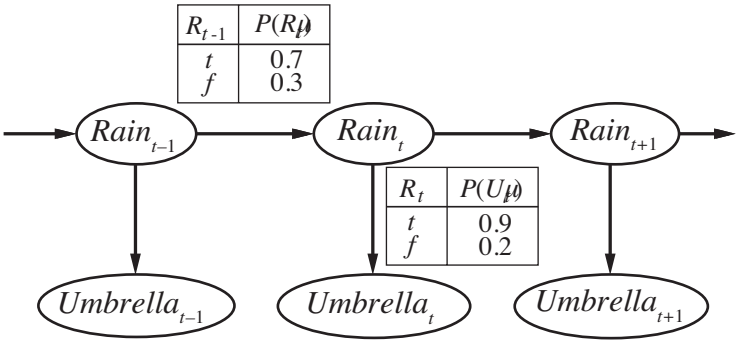


Figure 15.2 Bayesian network structure and conditional distributions describing the umbrella world. The transition model is $P(Rain_t \mid Rain_{t-1})$ and the sensor model is $P(Umbrella_t \mid Rain_t)$.

direction of the dependence between state and sensors: the arrows go from the actual state of the world to sensor values because the state of the world *causes* the sensors to take on particular values: the rain *causes* the umbrella to appear. (The inference process, of course, goes in the other direction; the distinction between the direction of modeled dependencies and the direction of inference is one of the principal advantages of Bayesian networks.)

In addition to specifying the transition and sensor models, we need to say how everything gets started—the prior probability distribution at time 0, $\mathbf{P}(\mathbf{X}_0)$. With that, we have a specification of the complete joint distribution over all the variables, using Equation (14.2). For any t,

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i \mid \mathbf{X}_{i-1}) \, \mathbf{P}(\mathbf{E}_i \mid \mathbf{X}_i) \; . \tag{15.3}$$

The three terms on the right-hand side are the initial state model $\mathbf{P}(\mathbf{X}_0)$, the transition model $\mathbf{P}(\mathbf{X}_i \mid \mathbf{X}_{i-1})$, and the sensor model $\mathbf{P}(\mathbf{E}_i \mid \mathbf{X}_i)$.

The structure in Figure 15.2 is a first-order Markov process—the probability of rain is assumed to depend only on whether it rained the previous day. Whether such an assumption is reasonable depends on the domain itself. The first-order Markov assumption says that the state variables contain *all* the information needed to characterize the probability distribution for the next time slice. Sometimes the assumption is exactly true—for example, if a particle is executing a random walk along the x-axis, changing its position by ± 1 at each time step, then using the x-coordinate as the state gives a first-order Markov process. Sometimes the assumption is only approximate, as in the case of predicting rain only on the basis of whether it rained the previous day. There are two ways to improve the accuracy of the approximation:

1. Increasing the order of the Markov process model. For example, we could make a second-order model by adding $Rain_{t-2}$ as a parent of $Rain_t$, which might give slightly more accurate predictions. For example, in Palo Alto, California, it very rarely rains more than two days in a row.
2. Increasing the set of state variables. For example, we could add $Season_t$ to allow

us to incorporate historical records of rainy seasons, or we could add $Temperature_t$, $Humidity_t$ and $Pressure_t$ (perhaps at a range of locations) to allow us to use a physical model of rainy conditions.

Exercise 15.1 asks you to show that the first solution—increasing the order—can always be reformulated as an increase in the set of state variables, keeping the order fixed. Notice that adding state variables might improve the system's predictive power but also increases the prediction *requirements*: we now have to predict the new variables as well. Thus, we are looking for a "self-sufficient" set of variables, which really means that we have to understand the "physics" of the process being modeled. The requirement for accurate modeling of the process is obviously lessened if we can add new sensors (e.g., measurements of temperature and pressure) that provide information directly about the new state variables.

Consider, for example, the problem of tracking a robot wandering randomly on the X–Y plane. One might propose that the position and velocity are a sufficient set of state variables: one can simply use Newton's laws to calculate the new position, and the velocity may change unpredictably. If the robot is battery-powered, however, then battery exhaustion would tend to have a systematic effect on the change in velocity. Because this in turn depends on how much power was used by all previous maneuvers, the Markov property is violated. We can restore the Markov property by including the charge level $Battery_t$ as one of the state variables that make up $\mathbf{X}_t$. This helps in predicting the motion of the robot, but in turn requires a model for predicting $Battery_t$ from $Battery_{t-1}$ and the velocity. In some cases, that can be done reliably, but more often we find that error accumulates over time. In that case, accuracy can be improved by *adding a new sensor* for the battery level.

15.2 Inference in Temporal Models

Having set up the structure of a generic temporal model, we can formulate the basic inference tasks that must be solved:

FILTERING
BELIEF STATE
STATE ESTIMATION

- **Filtering**: This is the task of computing the **belief state**—the posterior distribution over the most recent state—given all evidence to date. Filtering[2] is also called **state estimation**. In our example, we wish to compute $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$. In the umbrella example, this would mean computing the probability of rain today, given all the observations of the umbrella carrier made so far. Filtering is what a rational agent does to keep track of the current state so that rational decisions can be made. It turns out that an almost identical calculation provides the likelihood of the evidence sequence, $P(\mathbf{e}_{1:t})$.

PREDICTION

- **Prediction**: This is the task of computing the posterior distribution over the *future* state, given all evidence to date. That is, we wish to compute $\mathbf{P}(\mathbf{X}_{t+k} \mid \mathbf{e}_{1:t})$ for some $k > 0$. In the umbrella example, this might mean computing the probability of rain three days from now, given all the observations to date. Prediction is useful for evaluating possible courses of action based on their expected outcomes.

[2] The term "filtering" refers to the roots of this problem in early work on signal processing, where the problem is to filter out the noise in a signal by estimating its underlying properties.

SMOOTHING

- **Smoothing**: This is the task of computing the posterior distribution over a *past* state, given all evidence up to the present. That is, we wish to compute $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ for some k such that $0 \leq k < t$. In the umbrella example, it might mean computing the probability that it rained last Wednesday, given all the observations of the umbrella carrier made up to today. Smoothing provides a better estimate of the state than was available at the time, because it incorporates more evidence.[3]
- **Most likely explanation**: Given a sequence of observations, we might wish to find the sequence of states that is most likely to have generated those observations. That is, we wish to compute $\mathrm{argmax}_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t} \mid \mathbf{e}_{1:t})$. For example, if the umbrella appears on each of the first three days and is absent on the fourth, then the most likely explanation is that it rained on the first three days and did not rain on the fourth. Algorithms for this task are useful in many applications, including speech recognition—where the aim is to find the most likely sequence of words, given a series of sounds—and the reconstruction of bit strings transmitted over a noisy channel.

In addition to these inference tasks, we also have

- **Learning**: The transition and sensor models, if not yet known, can be learned from observations. Just as with static Bayesian networks, dynamic Bayes net learning can be done as a by-product of inference. Inference provides an estimate of what transitions actually occurred and of what states generated the sensor readings, and these estimates can be used to update the models. The updated models provide new estimates, and the process iterates to convergence. The overall process is an instance of the expectation-maximization or **EM algorithm**. (See Section 20.3.)

Note that learning requires smoothing, rather than filtering, because smoothing provides better estimates of the states of the process. Learning with filtering can fail to converge correctly; consider, for example, the problem of learning to solve murders: unless you are an eyewitness, smoothing is *always* required to infer what happened at the murder scene from the observable variables.

The remainder of this section describes generic algorithms for the four inference tasks, independent of the particular kind of model employed. Improvements specific to each model are described in subsequent sections.

15.2.1 Filtering and prediction

As we pointed out in Section 7.7.3, a useful filtering algorithm needs to maintain a current state estimate and update it, rather than going back over the entire history of percepts for each update. (Otherwise, the cost of each update increases as time goes by.) In other words, given the result of filtering up to time t, the agent needs to compute the result for $t+1$ from the new evidence $\mathbf{e}_{t+1}$,

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t}))\ ,$$

RECURSIVE ESTIMATION

for some function f. This process is called **recursive estimation**. We can view the calculation

[3] In particular, when tracking a moving object with inaccurate position observations, smoothing gives a smoother estimated trajectory than filtering—hence the name.

as being composed of two parts: first, the current state distribution is projected forward from t to $t+1$; then it is updated using the new evidence $\mathbf{e}_{t+1}$. This two-part process emerges quite simply when the formula is rearranged:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \quad \text{(dividing up the evidence)} \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}, \mathbf{e}_{1:t})\, \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \quad \text{(using Bayes' rule)} \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})\, \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \quad \text{(by the sensor Markov assumption).} \end{aligned} \tag{15.4}$$

Here and throughout this chapter, α is a normalizing constant used to make probabilities sum up to 1. The second term, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t})$ represents a one-step prediction of the next state, and the first term updates this with the new evidence; notice that $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})$ is obtainable directly from the sensor model. Now we obtain the one-step prediction for the next state by conditioning on the current state $\mathbf{X}_t$:

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\ &= \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \quad \text{(Markov assumption).} \end{aligned} \tag{15.5}$$

Within the summation, the first factor comes from the transition model and the second comes from the current state distribution. Hence, we have the desired recursive formulation. We can think of the filtered estimate $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ as a "message" $\mathbf{f}_{1:t}$ that is propagated forward along the sequence, modified by each transition and updated by each new observation. The process is given by

$$\mathbf{f}_{1:t+1} = \alpha\, \textsc{Forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1}) \ ,$$

where FORWARD implements the update described in Equation (15.5) and the process begins with $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0)$. When all the state variables are discrete, the time for each update is constant (i.e., independent of t), and the space required is also constant. (The constants depend, of course, on the size of the state space and the specific type of the temporal model in question.) *The time and space requirements for updating must be constant if an agent with limited memory is to keep track of the current state distribution over an unbounded sequence of observations.*

Let us illustrate the filtering process for two steps in the basic umbrella example (Figure 15.2.) That is, we will compute $\mathbf{P}(R_2 \mid u_{1:2})$ as follows:

- On day 0, we have no observations, only the security guard's prior beliefs; let's assume that consists of $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$.
- On day 1, the umbrella appears, so $U_1 = \mathit{true}$. The prediction from $t = 0$ to $t = 1$ is

$$\begin{aligned}\mathbf{P}(R_1) &= \sum_{r_0} \mathbf{P}(R_1 \mid r_0) P(r_0) \\ &= \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle \ . \end{aligned}$$

  Then the update step simply multiplies by the probability of the evidence for $t = 1$ and normalizes, as shown in Equation (15.4):

$$\begin{aligned}\mathbf{P}(R_1 \mid u_1) &= \alpha\, \mathbf{P}(u_1 \mid R_1) \mathbf{P}(R_1) = \alpha\, \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle \\ &= \alpha\, \langle 0.45, 0.1 \rangle \approx \langle 0.818, 0.182 \rangle \ . \end{aligned}$$

- On day 2, the umbrella appears, so $U_2 = true$. The prediction from $t = 1$ to $t = 2$ is

$$\begin{aligned}\mathbf{P}(R_2 \,|\, u_1) &= \sum_{r_1} \mathbf{P}(R_2 \,|\, r_1) P(r_1 \,|\, u_1) \\ &= \langle 0.7, 0.3\rangle \times 0.818 + \langle 0.3, 0.7\rangle \times 0.182 \approx \langle 0.627, 0.373\rangle \ ,\end{aligned}$$

and updating it with the evidence for $t = 2$ gives

$$\begin{aligned}\mathbf{P}(R_2 \,|\, u_1, u_2) &= \alpha\, \mathbf{P}(u_2 \,|\, R_2)\mathbf{P}(R_2 \,|\, u_1) = \alpha\, \langle 0.9, 0.2\rangle \langle 0.627, 0.373\rangle \\ &= \alpha\, \langle 0.565, 0.075\rangle \approx \langle 0.883, 0.117\rangle \ .\end{aligned}$$

Intuitively, the probability of rain increases from day 1 to day 2 because rain persists. Exercise 15.2(a) asks you to investigate this tendency further.

The task of **prediction** can be seen simply as filtering without the addition of new evidence. In fact, the filtering process already incorporates a one-step prediction, and it is easy to derive the following recursive computation for predicting the state at $t + k + 1$ from a prediction for $t + k$:

$$\mathbf{P}(\mathbf{X}_{t+k+1} \,|\, \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1} \,|\, \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} \,|\, \mathbf{e}_{1:t}) \ . \tag{15.6}$$

Naturally, this computation involves only the transition model and not the sensor model.

It is interesting to consider what happens as we try to predict further and further into the future. As Exercise 15.2(b) shows, the predicted distribution for rain converges to a fixed point $\langle 0.5, 0.5\rangle$, after which it remains constant for all time. This is the **stationary distribution** of the Markov process defined by the transition model. (See also page 537.) A great deal is known about the properties of such distributions and about the **mixing time**—roughly, the time taken to reach the fixed point. In practical terms, this dooms to failure any attempt to predict the *actual* state for a number of steps that is more than a small fraction of the mixing time, unless the stationary distribution itself is strongly peaked in a small area of the state space. The more uncertainty there is in the transition model, the shorter will be the mixing time and the more the future is obscured.

MIXING TIME

In addition to filtering and prediction, we can use a forward recursion to compute the **likelihood** of the evidence sequence, $P(\mathbf{e}_{1:t})$. This is a useful quantity if we want to compare different temporal models that might have produced the same evidence sequence (e.g., two different models for the persistence of rain). For this recursion, we use a likelihood message $\boldsymbol{\ell}_{1:t}(\mathbf{X}_t) = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$. It is a simple exercise to show that the message calculation is identical to that for filtering:

$$\boldsymbol{\ell}_{1:t+1} = \textsc{Forward}(\boldsymbol{\ell}_{1:t}, \mathbf{e}_{t+1}) \ .$$

Having computed $\boldsymbol{\ell}_{1:t}$, we obtain the actual likelihood by summing out $\mathbf{X}_t$:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \boldsymbol{\ell}_{1:t}(\mathbf{x}_t) \ . \tag{15.7}$$

Notice that the likelihood message represents the probabilities of longer and longer evidence sequences as time goes by and so becomes numerically smaller and smaller, leading to underflow problems with floating-point arithmetic. This is an important problem in practice, but we shall not go into solutions here.

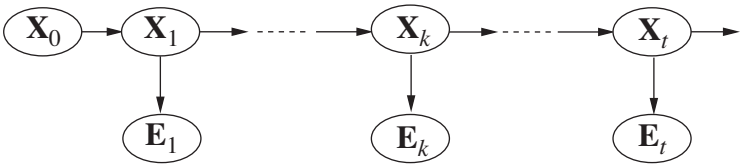


Figure 15.3 Smoothing computes $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$, the posterior distribution of the state at some past time k given a complete sequence of observations from 1 to t.

15.2.2 Smoothing

As we said earlier, smoothing is the process of computing the distribution over past states given evidence up to the present; that is, $\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t})$ for $0 \leq k < t$. (See Figure 15.3.) In anticipation of another recursive message-passing approach, we can split the computation into two parts—the evidence up to k and the evidence from $k+1$ to t,

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{(using Bayes' rule)} \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) \quad \text{(using conditional independence)} \\
&= \alpha\, \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t} \,. \qquad (15.8)
\end{aligned}$$

where "$\times$" represents pointwise multiplication of vectors. Here we have defined a "backward" message $\mathbf{b}_{k+1:t} = \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k)$, analogous to the forward message $\mathbf{f}_{1:k}$. The forward message $\mathbf{f}_{1:k}$ can be computed by filtering forward from 1 to k, as given by Equation (15.5). It turns out that the backward message $\mathbf{b}_{k+1:t}$ can be computed by a recursive process that runs *backward* from t:

$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(conditioning on } \mathbf{X}_{k+1}\text{)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(by conditional independence)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} \mid \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \,, \qquad (15.9)
\end{aligned}$$

where the last step follows by the conditional independence of $\mathbf{e}_{k+1}$ and $\mathbf{e}_{k+2:t}$, given $\mathbf{X}_{k+1}$. Of the three factors in this summation, the first and third are obtained directly from the model, and the second is the "recursive call." Using the message notation, we have

$$\mathbf{b}_{k+1:t} = \text{BACKWARD}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1}) \,,$$

where BACKWARD implements the update described in Equation (15.9). As with the forward recursion, the time and space needed for each update are constant and thus independent of t.

We can now see that the two terms in Equation (15.8) can both be computed by recursions through time, one running forward from 1 to k and using the filtering equation (15.5)

and the other running backward from t to $k+1$ and using Equation (15.9). Note that the backward phase is initialized with $\mathbf{b}_{t+1:t} = \mathbf{P}(\mathbf{e}_{t+1:t} \mid \mathbf{X}_t) = \mathbf{P}(\ \mid \mathbf{X}_t)\mathbf{1}$, where $\mathbf{1}$ is a vector of 1s. (Because $\mathbf{e}_{t+1:t}$ is an empty sequence, the probability of observing it is 1.)

Let us now apply this algorithm to the umbrella example, computing the smoothed estimate for the probability of rain at time $k=1$, given the umbrella observations on days 1 and 2. From Equation (15.8), this is given by

$$\mathbf{P}(R_1 \mid u_1, u_2) = \alpha\, \mathbf{P}(R_1 \mid u_1)\, \mathbf{P}(u_2 \mid R_1)\ . \tag{15.10}$$

The first term we already know to be $\langle .818, .182 \rangle$, from the forward filtering process described earlier. The second term can be computed by applying the backward recursion in Equation (15.9):

$$\begin{aligned}\mathbf{P}(u_2 \mid R_1) &= \sum_{r_2} P(u_2 \mid r_2) P(\ \mid r_2) \mathbf{P}(r_2 \mid R_1) \\ &= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle) = \langle 0.69, 0.41 \rangle\ .\end{aligned}$$

Plugging this into Equation (15.10), we find that the smoothed estimate for rain on day 1 is

$$\mathbf{P}(R_1 \mid u_1, u_2) = \alpha\, \langle 0.818, 0.182 \rangle \times \langle 0.69, 0.41 \rangle \approx\ \langle 0.883, 0.117 \rangle\ .$$

Thus, the smoothed estimate for rain on day 1 is *higher* than the filtered estimate (0.818) in this case. This is because the umbrella on day 2 makes it more likely to have rained on day 2; in turn, because rain tends to persist, that makes it more likely to have rained on day 1.

Both the forward and backward recursions take a constant amount of time per step; hence, the time complexity of smoothing with respect to evidence $\mathbf{e}_{1:t}$ is $O(t)$. This is the complexity for smoothing at a particular time step k. If we want to smooth the whole sequence, one obvious method is simply to run the whole smoothing process once for each time step to be smoothed. This results in a time complexity of $O(t^2)$. A better approach uses a simple application of dynamic programming to reduce the complexity to $O(t)$. A clue appears in the preceding analysis of the umbrella example, where we were able to reuse the results of the forward-filtering phase. The key to the linear-time algorithm is to *record the results* of forward filtering over the whole sequence. Then we run the backward recursion from t down to 1, computing the smoothed estimate at each step k from the computed backward message $\mathbf{b}_{k+1:t}$ and the stored forward message $\mathbf{f}_{1:k}$. The algorithm, aptly called the **forward–backward algorithm**, is shown in Figure 15.4.

FORWARD–BACKWARD ALGORITHM

The alert reader will have spotted that the Bayesian network structure shown in Figure 15.3 is a *polytree* as defined on page 528. This means that a straightforward application of the clustering algorithm also yields a linear-time algorithm that computes smoothed estimates for the entire sequence. It is now understood that the forward–backward algorithm is in fact a special case of the polytree propagation algorithm used with clustering methods (although the two were developed independently).

The forward–backward algorithm forms the computational backbone for many applications that deal with sequences of noisy observations. As described so far, it has two practical drawbacks. The first is that its space complexity can be too high when the state space is large and the sequences are long. It uses $O(|\mathbf{f}|t)$ space where $|\mathbf{f}|$ is the size of the representation of the forward message. The space requirement can be reduced to $O(|\mathbf{f}| \log t)$ with a concomi-

tant increase in the time complexity by a factor of $\log t$, as shown in Exercise 15.3. In some cases (see Section 15.3), a constant-space algorithm can be used.

The second drawback of the basic algorithm is that it needs to be modified to work in an *online* setting where smoothed estimates must be computed for earlier time slices as new observations are continuously added to the end of the sequence. The most common requirement is for **fixed-lag smoothing**, which requires computing the smoothed estimate $\mathbf{P}(\mathbf{X}_{t-d} \mid \mathbf{e}_{1:t})$ for fixed d. That is, smoothing is done for the time slice d steps behind the current time t; as t increases, the smoothing has to keep up. Obviously, we can run the forward–backward algorithm over the d-step "window" as each new observation is added, but this seems inefficient. In Section 15.3, we will see that fixed-lag smoothing can, in some cases, be done in constant time per update, independent of the lag d.

FIXED-LAG SMOOTHING

15.2.3 Finding the most likely sequence

Suppose that $[true, true, false, true, true]$ is the umbrella sequence for the security guard's first five days on the job. What is the weather sequence most likely to explain this? Does the absence of the umbrella on day 3 mean that it wasn't raining, or did the director forget to bring it? If it didn't rain on day 3, perhaps (because weather tends to persist) it didn't rain on day 4 either, but the director brought the umbrella just in case. In all, there are 2^5 possible weather sequences we could pick. Is there a way to find the most likely one, short of enumerating all of them?

We could try this linear-time procedure: use smoothing to find the posterior distribution for the weather at each time step; then construct the sequence, using at each step the weather that is most likely according to the posterior. Such an approach should set off alarm bells in the reader's head, because the posterior distributions computed by smoothing are distri-

function FORWARD-BACKWARD(**ev**, $prior$) **returns** a vector of probability distributions
  inputs: **ev**, a vector of evidence values for steps $1, \ldots, t$
    $prior$, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$
  local variables: **fv**, a vector of forward messages for steps $0, \ldots, t$
    b, a representation of the backward message, initially all 1s
    sv, a vector of smoothed estimates for steps $1, \ldots, t$

  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

Figure 15.4 The forward–backward algorithm for smoothing: computing posterior probabilities of a sequence of states given a sequence of observations. The FORWARD and BACKWARD operators are defined by Equations (15.5) and (15.9), respectively.

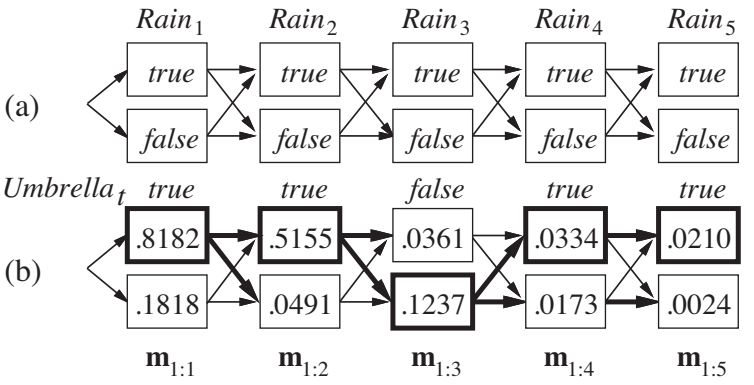


Figure 15.5 (a) Possible state sequences for $Rain_t$ can be viewed as paths through a graph of the possible states at each time step. (States are shown as rectangles to avoid confusion with nodes in a Bayes net.) (b) Operation of the Viterbi algorithm for the umbrella observation sequence $[true, true, false, true, true]$. For each t, we have shown the values of the message $\mathbf{m}_{1:t}$, which gives the probability of the best sequence reaching each state at time t. Also, for each state, the bold arrow leading into it indicates its best predecessor as measured by the product of the preceding sequence probability and the transition probability. Following the bold arrows back from the most likely state in $\mathbf{m}_{1:5}$ gives the most likely sequence.

butions over *single* time steps, whereas to find the most likely *sequence* we must consider *joint* probabilities over all the time steps. The results can in fact be quite different. (See Exercise 15.4.)

There *is* a linear-time algorithm for finding the most likely sequence, but it requires a little more thought. It relies on the same Markov property that yielded efficient algorithms for filtering and smoothing. The easiest way to think about the problem is to view each sequence as a *path* through a graph whose nodes are the possible *states* at each time step. Such a graph is shown for the umbrella world in Figure 15.5(a). Now consider the task of finding the most likely path through this graph, where the likelihood of any path is the product of the transition probabilities along the path and the probabilities of the given observations at each state. Let's focus in particular on paths that reach the state $Rain_5 = true$. Because of the Markov property, it follows that the most likely path to the state $Rain_5 = true$ consists of the most likely path to *some* state at time 4 followed by a transition to $Rain_5 = true$; and the state at time 4 that will become part of the path to $Rain_5 = true$ is whichever maximizes the likelihood of that path. In other words, *there is a recursive relationship between most likely paths to each state* $\mathbf{x}_{t+1}$ *and most likely paths to each state* $\mathbf{x}_t$. We can write this relationship as an equation connecting the probabilities of the paths:

$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1} \,|\, \mathbf{e}_{1:t+1}) \\ = \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \,|\, \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1} \,|\, \mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t \,|\, \mathbf{e}_{1:t}) \right) . \quad (15.11)$$

Equation (15.11) is *identical* to the filtering equation (15.5) except that

1. The forward message $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ is replaced by the message

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{X}_t \mid \mathbf{e}_{1:t}) \ ,$$

   that is, the probabilities of the most likely path to each state $\mathbf{x}_t$; and

2. the summation over $\mathbf{x}_t$ in Equation (15.5) is replaced by the maximization over $\mathbf{x}_t$ in Equation (15.11).

Thus, the algorithm for computing the most likely sequence is similar to filtering: it runs forward along the sequence, computing the **m** message at each time step, using Equation (15.11). The progress of this computation is shown in Figure 15.5(b). At the end, it will have the probability for the most likely sequence reaching *each* of the final states. One can thus easily select the most likely sequence overall (the states outlined in bold). In order to identify the actual sequence, as opposed to just computing its probability, the algorithm will also need to record, for each state, the best state that leads to it; these are indicated by the bold arrows in Figure 15.5(b). The optimal sequence is identified by following these bold arrows backwards from the best final state.

VITERBI ALGORITHM

The algorithm we have just described is called the **Viterbi algorithm**, after its inventor. Like the filtering algorithm, its time complexity is linear in t, the length of the sequence. Unlike filtering, which uses constant space, its space requirement is also linear in t. This is because the Viterbi algorithm needs to keep the pointers that identify the best sequence leading to each state.

15.3 HIDDEN MARKOV MODELS

The preceding section developed algorithms for temporal probabilistic reasoning using a general framework that was independent of the specific form of the transition and sensor models. In this and the next two sections, we discuss more concrete models and applications that illustrate the power of the basic algorithms and in some cases allow further improvements.

HIDDEN MARKOV MODEL

We begin with the **hidden Markov model**, or **HMM**. An HMM is a temporal probabilistic model in which the state of the process is described by a *single discrete* random variable. The possible values of the variable are the possible states of the world. The umbrella example described in the preceding section is therefore an HMM, since it has just one state variable: $Rain_t$. What happens if you have a model with two or more state variables? You can still fit it into the HMM framework by combining the variables into a single "megavariable" whose values are all possible tuples of values of the individual state variables. We will see that the restricted structure of HMMs allows for a simple and elegant matrix implementation of all the basic algorithms.[4]

[4] The reader unfamiliar with basic operations on vectors and matrices might wish to consult Appendix A before proceeding with this section.

15.3.1 Simplified matrix algorithms

With a single, discrete state variable X_t, we can give concrete form to the representations of the transition model, the sensor model, and the forward and backward messages. Let the state variable X_t have values denoted by integers $1, \ldots, S$, where S is the number of possible states. The transition model $\mathbf{P}(X_t \mid X_{t-1})$ becomes an $S \times S$ matrix $\mathbf{T}$, where

$$\mathbf{T}_{ij} = P(X_t = j \mid X_{t-1} = i) \ .$$

That is, $\mathbf{T}_{ij}$ is the probability of a transition from state i to state j. For example, the transition matrix for the umbrella world is

$$\mathbf{T} = \mathbf{P}(X_t \mid X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix} \ .$$

We also put the sensor model in matrix form. In this case, because the value of the evidence variable E_t is known at time t (call it e_t), we need only specify, for each state, how likely it is that the state causes e_t to appear: we need $P(e_t \mid X_t = i)$ for each state i. For mathematical convenience we place these values into an $S \times S$ diagonal matrix, $\mathbf{O}_t$ whose ith diagonal entry is $P(e_t \mid X_t = i)$ and whose other entries are 0. For example, on day 1 in the umbrella world of Figure 15.5, $U_1 = \mathit{true}$, and on day 3, $U_3 = \mathit{false}$, so, from Figure 15.2, we have

$$\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}; \qquad \mathbf{O}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix} \ .$$

Now, if we use column vectors to represent the forward and backward messages, all the computations become simple matrix–vector operations. The forward equation (15.5) becomes

$$\mathbf{f}_{1:t+1} = \alpha \, \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \tag{15.12}$$

and the backward equation (15.9) becomes

$$\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t} \ . \tag{15.13}$$

From these equations, we can see that the time complexity of the forward–backward algorithm (Figure 15.4) applied to a sequence of length t is $O(S^2 t)$, because each step requires multiplying an S-element vector by an $S \times S$ matrix. The space requirement is $O(St)$, because the forward pass stores t vectors of size S.

Besides providing an elegant description of the filtering and smoothing algorithms for HMMs, the matrix formulation reveals opportunities for improved algorithms. The first is a simple variation on the forward–backward algorithm that allows smoothing to be carried out in *constant* space, independently of the length of the sequence. The idea is that smoothing for any particular time slice k requires the simultaneous presence of both the forward and backward messages, $\mathbf{f}_{1:k}$ and $\mathbf{b}_{k+1:t}$, according to Equation (15.8). The forward–backward algorithm achieves this by storing the $\mathbf{f}$s computed on the forward pass so that they are available during the backward pass. Another way to achieve this is with a single pass that propagates both $\mathbf{f}$ and $\mathbf{b}$ in the same direction. For example, the "forward" message $\mathbf{f}$ can be propagated backward if we manipulate Equation (15.12) to work in the other direction:

$$\mathbf{f}_{1:t} = \alpha'(\mathbf{T}^\top)^{-1}\mathbf{O}_{t+1}^{-1}\mathbf{f}_{1:t+1} \ .$$

The modified smoothing algorithm works by first running the standard forward pass to compute $\mathbf{f}_{t:t}$ (forgetting all the intermediate results) and then running the backward pass for both

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 $\mathbf{P}(X_t|e_{1:t})$, 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: $\mathbf{O}_{t-d}, \mathbf{O}_t$, diagonal matrices containing the sensor model information

  add e_t to the end of $e_{t-d:t}$
  $\mathbf{O}_t \leftarrow$ diagonal matrix containing $\mathbf{P}(e_t|X_t)$
  if $t > d$ **then**
    $\mathbf{f} \leftarrow$ FORWARD($\mathbf{f}, e_t$)
    remove e_{t-d-1} from the beginning of $e_{t-d:t}$
    $\mathbf{O}_{t-d} \leftarrow$ diagonal matrix containing $\mathbf{P}(e_{t-d}|X_{t-d})$
    $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1}\mathbf{T}^{-1}\mathbf{B}\mathbf{T}\mathbf{O}_t$
  else $\mathbf{B} \leftarrow \mathbf{B}\mathbf{T}\mathbf{O}_t$
  $t \leftarrow t+1$
  if $t > d$ **then return** NORMALIZE($\mathbf{f} \times \mathbf{B1}$) **else return** null

Figure 15.6 An algorithm for smoothing with a fixed time lag of d steps, implemented as an online algorithm that outputs the new smoothed estimate given the observation for a new time step. Notice that the final output NORMALIZE($\mathbf{f} \times \mathbf{B1}$) is just $\alpha\,\mathbf{f} \times \mathbf{b}$, by Equation (15.14).

b and **f** together, using them to compute the smoothed estimate at each step. Since only one copy of each message is needed, the storage requirements are constant (i.e., independent of t, the length of the sequence). There are two significant restrictions on this algorithm: it requires that the transition matrix be invertible and that the sensor model have no zeroes—that is, that every observation be possible in every state.

A second area in which the matrix formulation reveals an improvement is in *online* smoothing with a fixed lag. The fact that smoothing can be done in constant space suggests that there should exist an efficient recursive algorithm for online smoothing—that is, an algorithm whose time complexity is independent of the length of the lag. Let us suppose that the lag is d; that is, we are smoothing at time slice $t-d$, where the current time is t. By Equation (15.8), we need to compute

$$\alpha\,\mathbf{f}_{1:t-d} \times \mathbf{b}_{t-d+1:t}$$

for slice $t-d$. Then, when a new observation arrives, we need to compute

$$\alpha\,\mathbf{f}_{1:t-d+1} \times \mathbf{b}_{t-d+2:t+1}$$

for slice $t-d+1$. How can this be done incrementally? First, we can compute $\mathbf{f}_{1:t-d+1}$ from $\mathbf{f}_{1:t-d}$, using the standard filtering process, Equation (15.5).

Computing the backward message incrementally is trickier, because there is no simple relationship between the old backward message $\mathbf{b}_{t-d+1:t}$ and the new backward message $\mathbf{b}_{t-d+2:t+1}$. Instead, we will examine the relationship between the old backward message $\mathbf{b}_{t-d+1:t}$ and the backward message at the front of the sequence, $\mathbf{b}_{t+1:t}$. To do this, we apply Equation (15.13) d times to get

$$\mathbf{b}_{t-d+1:t} = \left(\prod_{i=t-d+1}^{t} \mathbf{TO}_i \right) \mathbf{b}_{t+1:t} = \mathbf{B}_{t-d+1:t}\mathbf{1} \ , \tag{15.14}$$

where the matrix $\mathbf{B}_{t-d+1:t}$ is the product of the sequence of $\mathbf{T}$ and $\mathbf{O}$ matrices. $\mathbf{B}$ can be thought of as a "transformation operator" that transforms a later backward message into an earlier one. A similar equation holds for the new backward messages *after* the next observation arrives:

$$\mathbf{b}_{t-d+2:t+1} = \left(\prod_{i=t-d+2}^{t+1} \mathbf{TO}_i \right) \mathbf{b}_{t+2:t+1} = \mathbf{B}_{t-d+2:t+1}\mathbf{1} \ . \tag{15.15}$$

Examining the product expressions in Equations (15.14) and (15.15), we see that they have a simple relationship: to get the second product, "divide" the first product by the first element $\mathbf{TO}_{t-d+1}$, and multiply by the new last element $\mathbf{TO}_{t+1}$. In matrix language, then, there is a simple relationship between the old and new $\mathbf{B}$ matrices:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1}\mathbf{T}^{-1}\mathbf{B}_{t-d+1:t}\mathbf{T}\mathbf{O}_{t+1} \ . \tag{15.16}$$

This equation provides an incremental update for the $\mathbf{B}$ matrix, which in turn (through Equation (15.15)) allows us to compute the new backward message $\mathbf{b}_{t-d+2:t+1}$. The complete algorithm, which requires storing and updating $\mathbf{f}$ and $\mathbf{B}$, is shown in Figure 15.6.

15.3.2 Hidden Markov model example: Localization

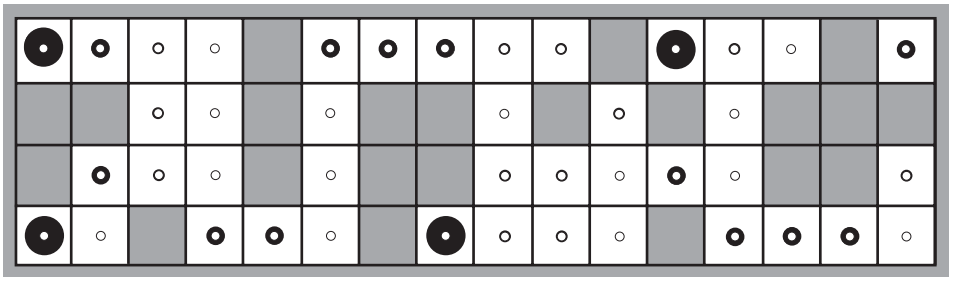
On page 145, we introduced a simple form of the **localization** problem for the vacuum world. In that version, the robot had a single nondeterministic *Move* action and its sensors reported perfectly whether or not obstacles lay immediately to the north, south, east, and west; the robot's belief state was the set of possible locations it could be in.

Here we make the problem slightly more realistic by including a simple probability model for the robot's motion and by allowing for noise in the sensors. The state variable X_t represents the location of the robot on the discrete grid; the domain of this variable is the set of empty squares $\{s_1, \ldots, s_n\}$. Let NEIGHBORS(s) be the set of empty squares that are adjacent to s and let $N(s)$ be the size of that set. Then the transition model for *Move* action says that the robot is equally likely to end up at any neighboring square:

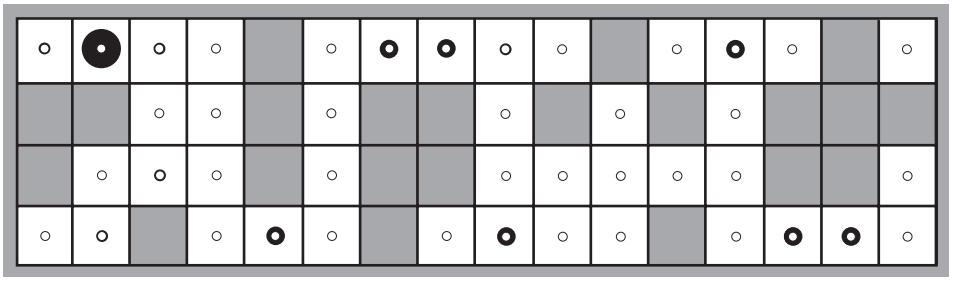
$$P(X_{t+1} = j \mid X_t = i) = \mathbf{T}_{ij} = (1/N(i) \text{ if } j \in \text{NEIGHBORS}(i) \text{ else } 0) \ .$$

We don't know where the robot starts, so we will assume a uniform distribution over all the squares; that is, $P(X_0 = i) = 1/n$. For the particular environment we consider (Figure 15.7), $n = 42$ and the transition matrix $\mathbf{T}$ has $42 \times 42 = 1764$ entries.

The sensor variable E_t has 16 possible values, each a four-bit sequence giving the presence or absence of an obstacle in a particular compass direction. We will use the notation



(a) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}$



(b) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}, \mathsf{E}_2 = \mathsf{NS}$

Figure 15.7 Posterior distribution over robot location: (a) one observation $E_1 = NSW$; (b) after a second observation $E_2 = NS$. The size of each disk corresponds to the probability that the robot is at that location. The sensor error rate is $\epsilon = 0.2$.

NS, for example, to mean that the north and south sensors report an obstacle and the east and west do not. Suppose that each sensor's error rate is ϵ and that errors occur independently for the four sensor directions. In that case, the probability of getting all four bits right is $(1-\epsilon)^4$ and the probability of getting them all wrong is ϵ^4. Furthermore, if d_{it} is the discrepancy—the number of bits that are different—between the true values for square i and the actual reading e_t, then the probability that a robot in square i would receive a sensor reading e_t is

$$P(E_t = e_t \mid X_t = i) = \mathbf{O}_{t_{ii}} = (1-\epsilon)^{4-d_{it}} \epsilon^{d_{it}} \ .$$

For example, the probability that a square with obstacles to the north and south would produce a sensor reading NSE is $(1-\epsilon)^3\epsilon^1$.

Given the matrices $\mathbf{T}$ and $\mathbf{O}_t$, the robot can use Equation (15.12) to compute the posterior distribution over locations—that is, to work out where it is. Figure 15.7 shows the distributions $\mathbf{P}(X_1 \mid E_1 = NSW)$ and $\mathbf{P}(X_2 \mid E_1 = NSW, E_2 = NS)$. This is the same maze we saw before in Figure 4.18 (page 146), but there we used logical filtering to find the locations that were *possible*, assuming perfect sensing. Those same locations are still the most *likely* with noisy sensing, but now *every* location has some nonzero probability.

In addition to filtering to estimate its current location, the robot can use smoothing (Equation (15.13)) to work out where it was at any given past time—for example, where it began at time 0—and it can use the Viterbi algorithm to work out the most likely path it has

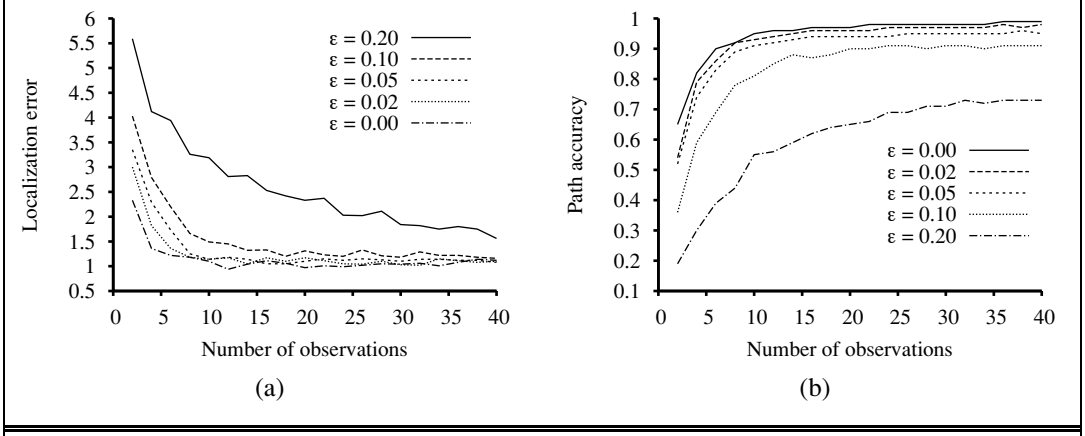


Figure 15.8 Performance of HMM localization as a function of the length of the observation sequence for various different values of the sensor error probability ϵ; data averaged over 400 runs. (a) The localization error, defined as the Manhattan distance from the true location. (b) The Viterbi path accuracy, defined as the fraction of correct states on the Viterbi path.

taken to get where it is now. Figure 15.8 shows the localization error and Viterbi path accuracy for various values of the per-bit sensor error rate ϵ. Even when ϵ is 20%—which means that the overall sensor reading is wrong 59% of the time—the robot is usually able to work out its location within two squares after 25 observations. This is because of the algorithm's ability to integrate evidence over time and to take into account the probabilistic constraints imposed on the location sequence by the transition model. When ϵ is 10%, the performance after a half-dozen observations is hard to distinguish from the performance with perfect sensing. Exercise 15.7 asks you to explore how robust the HMM localization algorithm is to errors in the prior distribution $\mathbf{P}(X_0)$ and in the transition model itself. Broadly speaking, high levels of localization and path accuracy are maintained even in the face of substantial errors in the models used.

The state variable for the example we have considered in this section is a physical location in the world. Other problems can, of course, include other aspects of the world. Exercise 15.8 asks you to consider a version of the vacuum robot that has the policy of going straight for as long as it can; only when it encounters an obstacle does it change to a new (randomly selected) heading. To model this robot, each state in the model consists of a *(location, heading)* pair. For the environment in Figure 15.7, which has 42 empty squares, this leads to 168 states and a transition matrix with $168^2 = 28,224$ entries—still a manageable number. If we add the possibility of dirt in the squares, the number of states is multiplied by 2^{42} and the transition matrix ends up with more than 10^{29} entries—no longer a manageable number; Section 15.5 shows how to use dynamic Bayesian networks to model domains with many state variables. If we allow the robot to move continuously rather than in a discrete grid, the number of states becomes infinite; the next section shows how to handle this case.

15.4 KALMAN FILTERS

Imagine watching a small bird flying through dense jungle foliage at dusk: you glimpse brief, intermittent flashes of motion; you try hard to guess where the bird is and where it will appear next so that you don't lose it. Or imagine that you are a World War II radar operator peering at a faint, wandering blip that appears once every 10 seconds on the screen. Or, going back further still, imagine you are Kepler trying to reconstruct the motions of the planets from a collection of highly inaccurate angular observations taken at irregular and imprecisely measured intervals. In all these cases, you are doing filtering: estimating state variables (here, position and velocity) from noisy observations over time. If the variables were discrete, we could model the system with a hidden Markov model. This section examines methods for handling continuous variables, using an algorithm called **Kalman filtering**, after one of its inventors, Rudolf E. Kalman.

KALMAN FILTERING

The bird's flight might be specified by six continuous variables at each time point; three for position (X_t, Y_t, Z_t) and three for velocity $(\dot{X}_t, \dot{Y}_t, \dot{Z}_t)$. We will need suitable conditional densities to represent the transition and sensor models; as in Chapter 14, we will use **linear Gaussian** distributions. This means that the next state $\mathbf{X}_{t+1}$ must be a linear function of the current state $\mathbf{X}_t$, plus some Gaussian noise, a condition that turns out to be quite reasonable in practice. Consider, for example, the X-coordinate of the bird, ignoring the other coordinates for now. Let the time interval between observations be Δ, and assume constant velocity during the interval; then the position update is given by $X_{t+\Delta} = X_t + \dot{X}\,\Delta$. Adding Gaussian noise (to account for wind variation, etc.), we obtain a linear Gaussian transition model:

$$P(X_{t+\Delta} = x_{t+\Delta} \mid X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t\,\Delta, \sigma^2)(x_{t+\Delta})\ .$$

The Bayesian network structure for a system with position vector $\mathbf{X}_t$ and velocity $\dot{\mathbf{X}}_t$ is shown in Figure 15.9. Note that this is a very specific form of linear Gaussian model; the general form will be described later in this section and covers a vast array of applications beyond the simple motion examples of the first paragraph. The reader might wish to consult Appendix A for some of the mathematical properties of Gaussian distributions; for our immediate purposes, the most important is that a **multivariate Gaussian** distribution for d variables is specified by a d-element mean $\boldsymbol{\mu}$ and a $d \times d$ covariance matrix $\boldsymbol{\Sigma}$.

MULTIVARIATE GAUSSIAN

15.4.1 Updating Gaussian distributions

In Chapter 14 on page 521, we alluded to a key property of the linear Gaussian family of distributions: it remains closed under the standard Bayesian network operations. Here, we make this claim precise in the context of filtering in a temporal probability model. The required properties correspond to the two-step filtering calculation in Equation (15.5):

1. If the current distribution $\mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ is Gaussian and the transition model $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$ is linear Gaussian, then the one-step predicted distribution given by

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t})\, d\mathbf{x}_t \tag{15.17}$$

is also a Gaussian distribution.

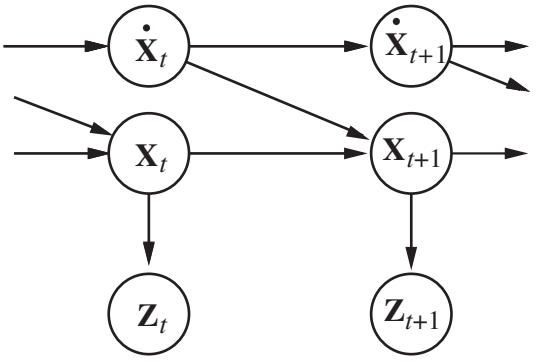


Figure 15.9 Bayesian network structure for a linear dynamical system with position $\mathbf{X}_t$, velocity $\dot{\mathbf{X}}_t$, and position measurement $\mathbf{Z}_t$.

2. If the prediction $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t})$ is Gaussian and the sensor model $\mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})$ is linear Gaussian, then, after conditioning on the new evidence, the updated distribution

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = \alpha\, \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \tag{15.18}$$

is also a Gaussian distribution.

Thus, the FORWARD operator for Kalman filtering takes a Gaussian forward message $\mathbf{f}_{1:t}$, specified by a mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$, and produces a new multivariate Gaussian forward message $\mathbf{f}_{1:t+1}$, specified by a mean $\boldsymbol{\mu}_{t+1}$ and covariance matrix $\boldsymbol{\Sigma}_{t+1}$. So, if we start with a Gaussian prior $\mathbf{f}_{1:0} = \mathbf{P}(\mathbf{X}_0) = N(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, filtering with a linear Gaussian model produces a Gaussian state distribution for all time.

This seems to be a nice, elegant result, but why is it so important? The reason is that, except for a few special cases such as this, *filtering with continuous or hybrid (discrete and continuous) networks generates state distributions whose representation grows without bound over time.* This statement is not easy to prove in general, but Exercise 15.10 shows what happens for a simple example.

15.4.2 A simple one-dimensional example

We have said that the FORWARD operator for the Kalman filter maps a Gaussian into a new Gaussian. This translates into computing a new mean and covariance matrix from the previous mean and covariance matrix. Deriving the update rule in the general (multivariate) case requires rather a lot of linear algebra, so we will stick to a very simple univariate case for now; and later give the results for the general case. Even for the univariate case, the calculations are somewhat tedious, but we feel that they are worth seeing because the usefulness of the Kalman filter is tied so intimately to the mathematical properties of Gaussian distributions.

The temporal model we consider describes a **random walk** of a single continuous state variable X_t with a noisy observation Z_t. An example might be the "consumer confidence" index, which can be modeled as undergoing a random Gaussian-distributed change each month and is measured by a random consumer survey that also introduces Gaussian sampling noise.

The prior distribution is assumed to be Gaussian with variance σ_0^2:

$$P(x_0) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} .$$

(For simplicity, we use the same symbol α for all normalizing constants in this section.) The transition model adds a Gaussian perturbation of constant variance σ_x^2 to the current state:

$$P(x_{t+1} \,|\, x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_{t+1}-x_t)^2}{\sigma_x^2}\right)} .$$

The sensor model assumes Gaussian noise with variance σ_z^2:

$$P(z_t \,|\, x_t) = \alpha\, e^{-\frac{1}{2}\left(\frac{(z_t-x_t)^2}{\sigma_z^2}\right)} .$$

Now, given the prior $\mathbf{P}(X_0)$, the one-step predicted distribution comes from Equation (15.17):

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1 \,|\, x_0) P(x_0)\, dx_0 = \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(x_1-x_0)^2}{\sigma_x^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_0-\mu_0)^2}{\sigma_0^2}\right)} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{\sigma_0^2(x_1-x_0)^2+\sigma_x^2(x_0-\mu_0)^2}{\sigma_0^2\sigma_x^2}\right)} dx_0 . \end{aligned}$$

This integral looks rather complicated. The key to progress is to notice that the exponent is the sum of two expressions that are *quadratic* in x_0 and hence is itself a quadratic in x_0. A simple trick known as **completing the square** allows the rewriting of any quadratic $ax_0^2 + bx_0 + c$ as the sum of a squared term $a(x_0 - \frac{-b}{2a})^2$ and a residual term $c - \frac{b^2}{4a}$ that is independent of x_0. The residual term can be taken outside the integral, giving us

COMPLETING THE SQUARE

$$P(x_1) = \alpha\, e^{-\frac{1}{2}\left(c-\frac{b^2}{4a}\right)} \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(a(x_0-\frac{-b}{2a})^2\right)} dx_0 .$$

Now the integral is just the integral of a Gaussian over its full range, which is simply 1. Thus, we are left with only the residual term from the quadratic. Then, we notice that the residual term is a quadratic in x_1; in fact, after simplification, we obtain

$$P(x_1) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)} .$$

That is, the one-step predicted distribution is a Gaussian with the same mean μ_0 and a variance equal to the sum of the original variance σ_0^2 and the transition variance σ_x^2.

To complete the update step, we need to condition on the observation at the first time step, namely, z_1. From Equation (15.18), this is given by

$$\begin{aligned} P(x_1 \,|\, z_1) &= \alpha\, P(z_1 \,|\, x_1) P(x_1) \\ &= \alpha\, e^{-\frac{1}{2}\left(\frac{(z_1-x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)} . \end{aligned}$$

Once again, we combine the exponents and complete the square (Exercise 15.11), obtaining

$$P(x_1 \,|\, z_1) = \alpha\, e^{-\frac{1}{2}\left(\frac{(x_1-\frac{(\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})^2}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)} . \tag{15.19}$$

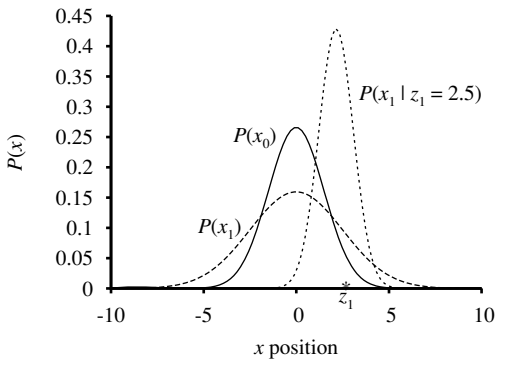


Figure 15.10 Stages in the Kalman filter update cycle for a random walk with a prior given by $\mu_0 = 0.0$ and $\sigma_0 = 1.0$, transition noise given by $\sigma_x = 2.0$, sensor noise given by $\sigma_z = 1.0$, and a first observation $z_1 = 2.5$ (marked on the x-axis). Notice how the prediction $P(x_1)$ is flattened out, relative to $P(x_0)$, by the transition noise. Notice also that the mean of the posterior distribution $P(x_1 \mid z_1)$ is slightly to the left of the observation z_1 because the mean is a weighted average of the prediction and the observation.

Thus, after one update cycle, we have a new Gaussian distribution for the state variable.

From the Gaussian formula in Equation (15.19), we see that the new mean and standard deviation can be calculated from the old mean and standard deviation as follows:

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \qquad \text{and} \qquad \sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \,. \tag{15.20}$$

Figure 15.10 shows one update cycle for particular values of the transition and sensor models.

Equation (15.20) plays exactly the same role as the general filtering equation (15.5) or the HMM filtering equation (15.12). Because of the special nature of Gaussian distributions, however, the equations have some interesting additional properties. First, we can interpret the calculation for the new mean μ_{t+1} as simply a *weighted mean* of the new observation z_{t+1} and the old mean μ_t. If the observation is unreliable, then σ_z^2 is large and we pay more attention to the old mean; if the old mean is unreliable (σ_t^2 is large) or the process is highly unpredictable (σ_x^2 is large), then we pay more attention to the observation. Second, notice that the update for the variance σ_{t+1}^2 is *independent of the observation*. We can therefore compute in advance what the sequence of variance values will be. Third, the sequence of variance values converges quickly to a fixed value that depends only on σ_x^2 and σ_z^2, thereby substantially simplifying the subsequent calculations. (See Exercise 15.12.)

15.4.3 The general case

The preceding derivation illustrates the key property of Gaussian distributions that allows Kalman filtering to work: the fact that the exponent is a quadratic form. This is true not just for the univariate case; the full multivariate Gaussian distribution has the form

$$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})(\mathbf{x}) = \alpha \, e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right)} \,.$$

Multiplying out the terms in the exponent makes it clear that the exponent is also a quadratic function of the values x_i in $\mathbf{x}$. As in the univariate case, the filtering update preserves the Gaussian nature of the state distribution.

Let us first define the general temporal model used with Kalman filtering. Both the transition model and the sensor model allow for a *linear* transformation with additive Gaussian noise. Thus, we have

$$\begin{aligned} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \mathbf{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t \mid \mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \mathbf{\Sigma}_z)(\mathbf{z}_t) \,, \end{aligned} \tag{15.21}$$

where $\mathbf{F}$ and $\mathbf{\Sigma}_x$ are matrices describing the linear transition model and transition noise covariance, and $\mathbf{H}$ and $\mathbf{\Sigma}_z$ are the corresponding matrices for the sensor model. Now the update equations for the mean and covariance, in their full, hairy horribleness, are

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \mathbf{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x) \,, \end{aligned} \tag{15.22}$$

where $\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top + \mathbf{\Sigma}_z)^{-1}$ is called the **Kalman gain matrix**. Believe it or not, these equations make some intuitive sense. For example, consider the update for the mean state estimate $\boldsymbol{\mu}$. The term $\mathbf{F}\boldsymbol{\mu}_t$ is the *predicted* state at $t+1$, so $\mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ is the *predicted* observation. Therefore, the term $\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$ represents the error in the predicted observation. This is multiplied by $\mathbf{K}_{t+1}$ to correct the predicted state; hence, $\mathbf{K}_{t+1}$ is a measure of *how seriously to take the new observation* relative to the prediction. As in Equation (15.20), we also have the property that the variance update is independent of the observations. The sequence of values for $\mathbf{\Sigma}_t$ and $\mathbf{K}_t$ can therefore be computed offline, and the actual calculations required during online tracking are quite modest.

KALMAN GAIN MATRIX

To illustrate these equations at work, we have applied them to the problem of tracking an object moving on the X–Y plane. The state variables are $\mathbf{X} = (X, Y, \dot{X}, \dot{Y})^\top$, so $\mathbf{F}$, $\mathbf{\Sigma}_x$, $\mathbf{H}$, and $\mathbf{\Sigma}_z$ are 4×4 matrices. Figure 15.11(a) shows the true trajectory, a series of noisy observations, and the trajectory estimated by Kalman filtering, along with the covariances indicated by the one-standard-deviation contours. The filtering process does a good job of tracking the actual motion, and, as expected, the variance quickly reaches a fixed point.

We can also derive equations for *smoothing* as well as filtering with linear Gaussian models. The smoothing results are shown in Figure 15.11(b). Notice how the variance in the position estimate is sharply reduced, except at the ends of the trajectory (why?), and that the estimated trajectory is much smoother.

15.4.4 Applicability of Kalman filtering

The Kalman filter and its elaborations are used in a vast array of applications. The "classical" application is in radar tracking of aircraft and missiles. Related applications include acoustic tracking of submarines and ground vehicles and visual tracking of vehicles and people. In a slightly more esoteric vein, Kalman filters are used to reconstruct particle trajectories from bubble-chamber photographs and ocean currents from satellite surface measurements. The range of application is much larger than just the tracking of motion: any system characterized by continuous state variables and noisy measurements will do. Such systems include pulp mills, chemical plants, nuclear reactors, plant ecosystems, and national economies.

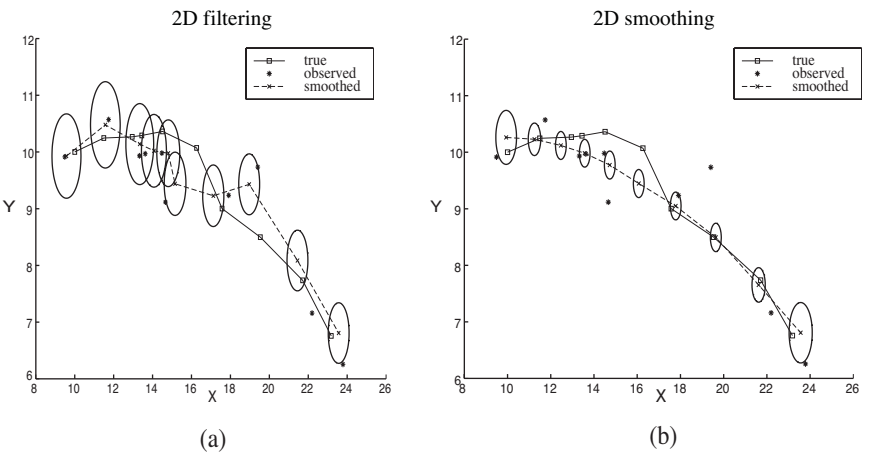


Figure 15.11 (a) Results of Kalman filtering for an object moving on the X–Y plane, showing the true trajectory (left to right), a series of noisy observations, and the trajectory estimated by Kalman filtering. Variance in the position estimate is indicated by the ovals. (b) The results of Kalman smoothing for the same observation sequence.

The fact that Kalman filtering can be applied to a system does not mean that the results will be valid or useful. The assumptions made—a linear Gaussian transition and sensor models—are very strong. The **extended Kalman filter (EKF)** attempts to overcome nonlinearities in the system being modeled. A system is **nonlinear** if the transition model cannot be described as a matrix multiplication of the state vector, as in Equation (15.21). The EKF works by modeling the system as *locally* linear in $\mathbf{x}_t$ in the region of $\mathbf{x}_t = \boldsymbol{\mu}_t$, the mean of the current state distribution. This works well for smooth, well-behaved systems and allows the tracker to maintain and update a Gaussian state distribution that is a reasonable approximation to the true posterior. A detailed example is given in Chapter 25.

EXTENDED KALMAN FILTER (EKF)

NONLINEAR

What does it mean for a system to be "unsmooth" or "poorly behaved"? Technically, it means that there is significant nonlinearity in system response within the region that is "close" (according to the covariance $\boldsymbol{\Sigma}_t$) to the current mean $\boldsymbol{\mu}_t$. To understand this idea in nontechnical terms, consider the example of trying to track a bird as it flies through the jungle. The bird appears to be heading at high speed straight for a tree trunk. The Kalman filter, whether regular or extended, can make only a Gaussian prediction of the location of the bird, and the mean of this Gaussian will be centered on the trunk, as shown in Figure 15.12(a). A reasonable model of the bird, on the other hand, would predict evasive action to one side or the other, as shown in Figure 15.12(b). Such a model is highly nonlinear, because the bird's decision varies sharply depending on its precise location relative to the trunk.

To handle examples like these, we clearly need a more expressive language for representing the behavior of the system being modeled. Within the control theory community, for which problems such as evasive maneuvering by aircraft raise the same kinds of difficulties, the standard solution is the **switching Kalman filter**. In this approach, multiple Kalman fil-

SWITCHING KALMAN FILTER

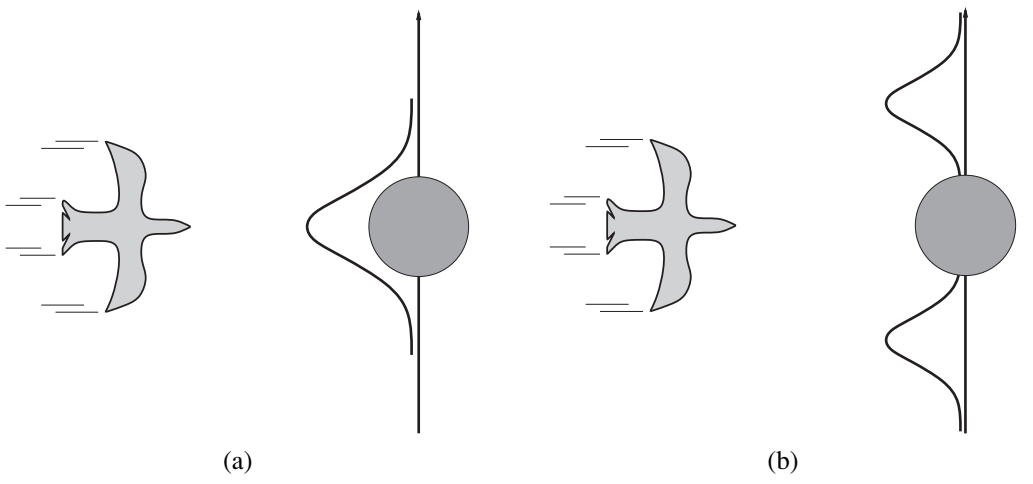


Figure 15.12 A bird flying toward a tree (top views). (a) A Kalman filter will predict the location of the bird using a single Gaussian centered on the obstacle. (b) A more realistic model allows for the bird's evasive action, predicting that it will fly to one side or the other.

ters run in parallel, each using a different model of the system—for example, one for straight flight, one for sharp left turns, and one for sharp right turns. A weighted sum of predictions is used, where the weight depends on how well each filter fits the current data. We will see in the next section that this is simply a special case of the general dynamic Bayesian network model, obtained by adding a discrete "maneuver" state variable to the network shown in Figure 15.9. Switching Kalman filters are discussed further in Exercise 15.10.

15.5 Dynamic Bayesian Networks

DYNAMIC BAYESIAN NETWORK

A **dynamic Bayesian network**, or **DBN**, is a Bayesian network that represents a temporal probability model of the kind described in Section 15.1. We have already seen examples of DBNs: the umbrella network in Figure 15.2 and the Kalman filter network in Figure 15.9. In general, each slice of a DBN can have any number of state variables $\mathbf{X}_t$ and evidence variables $\mathbf{E}_t$. For simplicity, we assume that the variables and their links are exactly replicated from slice to slice and that the DBN represents a first-order Markov process, so that each variable can have parents only in its own slice or the immediately preceding slice.

It should be clear that every hidden Markov model can be represented as a DBN with a single state variable and a single evidence variable. It is also the case that every discrete-variable DBN can be represented as an HMM; as explained in Section 15.3, we can combine all the state variables in the DBN into a single state variable whose values are all possible tuples of values of the individual state variables. Now, if every HMM is a DBN and every DBN can be translated into an HMM, what's the difference? The difference is that, *by de-*

composing the state of a complex system into its constituent variables, the can take advantage of sparseness *in the temporal probability model.* Suppose, for example, that a DBN has 20 Boolean state variables, each of which has three parents in the preceding slice. Then the DBN transition model has $20 \times 2^3 = 160$ probabilities, whereas the corresponding HMM has 2^{20} states and therefore 2^{40}, or roughly a trillion, probabilities in the transition matrix. This is bad for at least three reasons: first, the HMM itself requires much more space; second, the huge transition matrix makes HMM inference much more expensive; and third, the problem of learning such a huge number of parameters makes the pure HMM model unsuitable for large problems. The relationship between DBNs and HMMs is roughly analogous to the relationship between ordinary Bayesian networks and full tabulated joint distributions.

We have already explained that every Kalman filter model can be represented in a DBN with continuous variables and linear Gaussian conditional distributions (Figure 15.9). It should be clear from the discussion at the end of the preceding section that *not* every DBN can be represented by a Kalman filter model. In a Kalman filter, the current state distribution is always a single multivariate Gaussian distribution—that is, a single "bump" in a particular location. DBNs, on the other hand, can model arbitrary distributions. For many real-world applications, this flexibility is essential. Consider, for example, the current location of my keys. They might be in my pocket, on the bedside table, on the kitchen counter, dangling from the front door, or locked in the car. A single Gaussian bump that included all these places would have to allocate significant probability to the keys being in mid-air in the front hall. Aspects of the real world such as purposive agents, obstacles, and pockets introduce "nonlinearities" that require combinations of discrete and continuous variables in order to get reasonable models.

15.5.1 Constructing DBNs

To construct a DBN, one must specify three kinds of information: the prior distribution over the state variables, $\mathbf{P}(\mathbf{X}_0)$; the transition model $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t)$; and the sensor model $\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$. To specify the transition and sensor models, one must also specify the topology of the connections between successive slices and between the state and evidence variables. Because the transition and sensor models are assumed to be stationary—the same for all t—it is most convenient simply to specify them for the first slice. For example, the complete DBN specification for the umbrella world is given by the three-node network shown in Figure 15.13(a). From this specification, the complete DBN with an unbounded number of time slices can be constructed as needed by copying the first slice.

Let us now consider a more interesting example: monitoring a battery-powered robot moving in the X–Y plane, as introduced at the end of Section 15.1. First, we need state variables, which will include both $\mathbf{X}_t = (X_t, Y_t)$ for position and $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$ for velocity. We assume some method of measuring position—perhaps a fixed camera or onboard GPS (Global Positioning System)—yielding measurements $\mathbf{Z}_t$. The position at the next time step depends on the current position and velocity, as in the standard Kalman filter model. The velocity at the next step depends on the current velocity and the state of the battery. We add $Battery_t$ to represent the actual battery charge level, which has as parents the previous

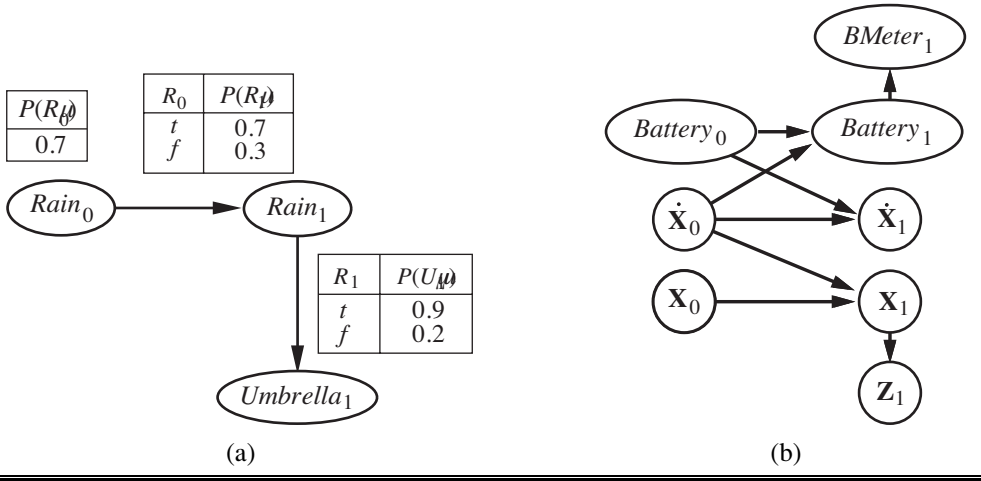


Figure 15.13 (a) Specification of the prior, transition model, and sensor model for the umbrella DBN. All subsequent slices are assumed to be copies of slice 1. (b) A simple DBN for robot motion in the X–Y plane.

battery level and the velocity, and we add $BMeter_t$, which measures the battery charge level. This gives us the basic model shown in Figure 15.13(b).

It is worth looking in more depth at the nature of the sensor model for $BMeter_t$. Let us suppose, for simplicity, that both $Battery_t$ and $BMeter_t$ can take on discrete values 0 through 5. If the meter is always accurate, then the CPT $\mathbf{P}(BMeter_t \mid Battery_t)$ should have probabilities of 1.0 "along the diagonal" and probabilities of 0.0 elsewhere. In reality, noise always creeps into measurements. For continuous measurements, a Gaussian distribution with a small variance might be used.[5] For our discrete variables, we can approximate a Gaussian using a distribution in which the probability of error drops off in the appropriate way, so that the probability of a large error is very small. We use the term **Gaussian error model** to cover both the continuous and discrete versions.

GAUSSIAN ERROR MODEL

Anyone with hands-on experience of robotics, computerized process control, or other forms of automatic sensing will readily testify to the fact that small amounts of measurement noise are often the least of one's problems. Real sensors *fail*. When a sensor fails, it does not necessarily send a signal saying, "Oh, by the way, the data I'm about to send you is a load of nonsense." Instead, it simply sends the nonsense. The simplest kind of failure is called a **transient failure**, where the sensor occasionally decides to send some nonsense. For example, the battery level sensor might have a habit of sending a zero when someone bumps the robot, even if the battery is fully charged.

TRANSIENT FAILURE

Let's see what happens when a transient failure occurs with a Gaussian error model that doesn't accommodate such failures. Suppose, for example, that the robot is sitting quietly and observes 20 consecutive battery readings of 5. Then the battery meter has a temporary seizure

[5] Strictly speaking, a Gaussian distribution is problematic because it assigns nonzero probability to large negative charge levels. The **beta distribution** is sometimes a better choice for a variable whose range is restricted.

and the next reading is $BMeter_{21} = 0$. What will the simple Gaussian error model lead us to believe about $Battery_{21}$? According to Bayes' rule, the answer depends on both the sensor model $\mathbf{P}(BMeter_{21} = 0 \mid Battery_{21})$ and the prediction $\mathbf{P}(Battery_{21} \mid BMeter_{1:20})$. If the probability of a large sensor error is significantly less likely than the probability of a transition to $Battery_{21} = 0$, even if the latter is very unlikely, then the posterior distribution will assign a high probability to the battery's being empty. A second reading of 0 at $t = 22$ will make this conclusion almost certain. If the transient failure then disappears and the reading returns to 5 from $t = 23$ onwards, the estimate for the battery level will quickly return to 5, as if by magic. This course of events is illustrated in the upper curve of Figure 15.14(a), which shows the expected value of $Battery_t$ over time, using a discrete Gaussian error model.

Despite the recovery, there is a time ($t = 22$) when the robot is convinced that its battery is empty; presumably, then, it should send out a mayday signal and shut down. Alas, its oversimplified sensor model has led it astray. How can this be fixed? Consider a familiar example from everyday human driving: on sharp curves or steep hills, one's "fuel tank empty" warning light sometimes turns on. Rather than looking for the emergency phone, one simply recalls that the fuel gauge sometimes gives a very large error when the fuel is sloshing around in the tank. The moral of the story is the following: *for the system to handle sensor failure properly, the sensor model must include the possibility of failure.*

The simplest kind of failure model for a sensor allows a certain probability that the sensor will return some completely incorrect value, regardless of the true state of the world. For example, if the battery meter fails by returning 0, we might say that

$$P(BMeter_t = 0 \mid Battery_t = 5) = 0.03 \ ,$$

TRANSIENT FAILURE MODEL

which is presumably much larger than the probability assigned by the simple Gaussian error model. Let's call this the **transient failure model**. How does it help when we are faced with a reading of 0? Provided that the *predicted* probability of an empty battery, according to the readings so far, is much less than 0.03, then the best explanation of the observation $BMeter_{21} = 0$ is that the sensor has temporarily failed. Intuitively, we can think of the belief about the battery level as having a certain amount of "inertia" that helps to overcome temporary blips in the meter reading. The upper curve in Figure 15.14(b) shows that the transient failure model can handle transient failures without a catastrophic change in beliefs.

So much for temporary blips. What about a persistent sensor failure? Sadly, failures of this kind are all too common. If the sensor returns 20 readings of 5 followed by 20 readings of 0, then the transient sensor failure model described in the preceding paragraph will result in the robot gradually coming to believe that its battery is empty when in fact it may be that the meter has failed. The lower curve in Figure 15.14(b) shows the belief "trajectory" for this case. By $t = 25$—five readings of 0—the robot is convinced that its battery is empty. Obviously, we would prefer the robot to believe that its battery meter is broken—if indeed this is the more likely event.

PERSISTENT FAILURE MODEL

Unsurprisingly, to handle persistent failure, we need a **persistent failure model** that describes how the sensor behaves under normal conditions and after failure. To do this, we need to augment the state of the system with an additional variable, say, $BMBroken$, that describes the status of the battery meter. The persistence of failure must be modeled by an

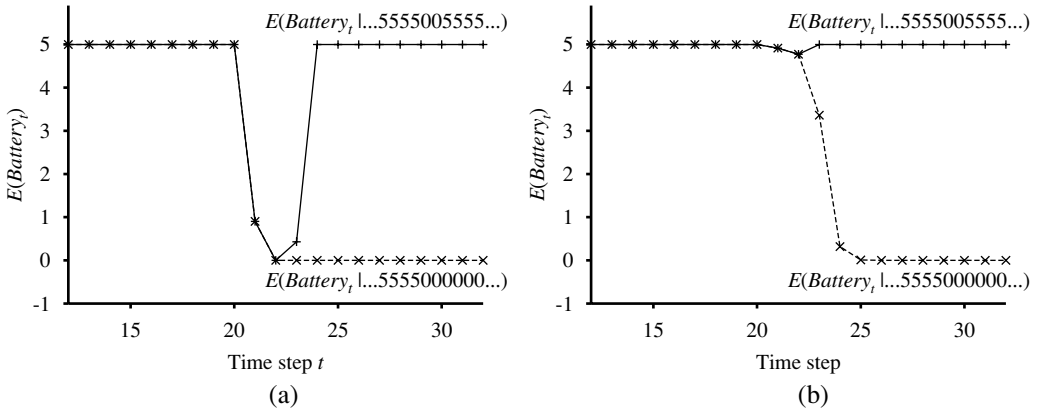


Figure 15.14 (a) Upper curve: trajectory of the expected value of $Battery_t$ for an observation sequence consisting of all 5s except for 0s at $t = 21$ and $t = 22$, using a simple Gaussian error model. Lower curve: trajectory when the observation remains at 0 from $t = 21$ onwards. (b) The same experiment run with the transient failure model. Notice that the transient failure is handled well, but the persistent failure results in excessive pessimism about the battery charge.

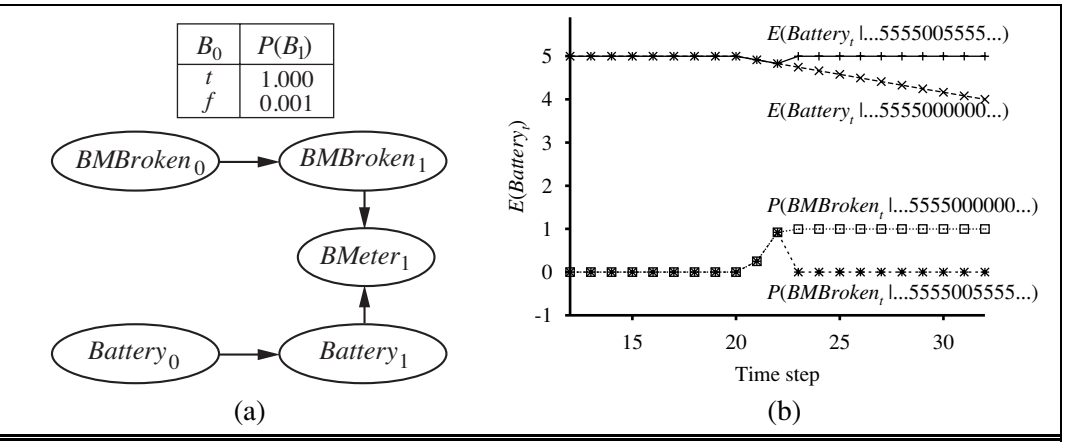


Figure 15.15 (a) A DBN fragment showing the sensor status variable required for modeling persistent failure of the battery sensor. (b) Upper curves: trajectories of the expected value of $Battery_t$ for the "transient failure" and "permanent failure" observations sequences. Lower curves: probability trajectories for $BMBroken$ given the two observation sequences.

PERSISTENCE ARC

arc linking $BMBroken_0$ to $BMBroken_1$. This **persistence arc** has a CPT that gives a small probability of failure in any given time step, say, 0.001, but specifies that the sensor stays broken once it breaks. When the sensor is OK, the sensor model for $BMeter$ is identical to the transient failure model; when the sensor is broken, it says $BMeter$ is always 0, regardless of the actual battery charge.

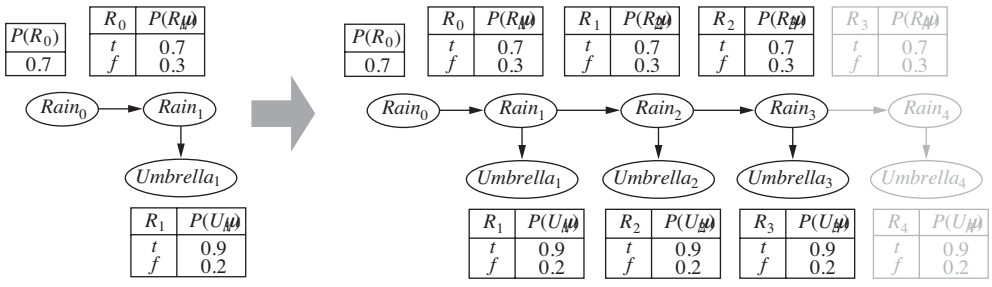


Figure 15.16 Unrolling a dynamic Bayesian network: slices are replicated to accommodate the observation sequence $Umbrella_{1:3}$. Further slices have no effect on inferences within the observation period.

The persistent failure model for the battery sensor is shown in Figure 15.15(a). Its performance on the two data sequences (temporary blip and persistent failure) is shown in Figure 15.15(b). There are several things to notice about these curves. First, in the case of the temporary blip, the probability that the sensor is broken rises significantly after the second 0 reading, but immediately drops back to zero once a 5 is observed. Second, in the case of persistent failure, the probability that the sensor is broken rises quickly to almost 1 and stays there. Finally, once the sensor is known to be broken, the robot can only assume that its battery discharges at the "normal" rate, as shown by the gradually descending level of $E(Battery_t \mid \ldots)$.

So far, we have merely scratched the surface of the problem of representing complex processes. The variety of transition models is huge, encompassing topics as disparate as modeling the human endocrine system and modeling multiple vehicles driving on a freeway. Sensor modeling is also a vast subfield in itself, but even subtle phenomena, such as sensor drift, sudden decalibration, and the effects of exogenous conditions (such as weather) on sensor readings, can be handled by explicit representation within dynamic Bayesian networks.

15.5.2 Exact inference in DBNs

Having sketched some ideas for representing complex processes as DBNs, we now turn to the question of inference. In a sense, this question has already been answered: dynamic Bayesian networks *are* Bayesian networks, and we already have algorithms for inference in Bayesian networks. Given a sequence of observations, one can construct the full Bayesian network representation of a DBN by replicating slices until the network is large enough to accommodate the observations, as in Figure 15.16. This technique, mentioned in Chapter 14 in the context of relational probability models, is called **unrolling**. (Technically, the DBN is equivalent to the semi-infinite network obtained by unrolling forever. Slices added beyond the last observation have no effect on inferences within the observation period and can be omitted.) Once the DBN is unrolled, one can use any of the inference algorithms—variable elimination, clustering methods, and so on—described in Chapter 14.

Unfortunately, a naive application of unrolling would not be particularly efficient. If we want to perform filtering or smoothing with a long sequence of observations $\mathbf{e}_{1:t}$, the

unrolled network would require $O(t)$ space and would thus grow without bound as more observations were added. Moreover, if we simply run the inference algorithm anew each time an observation is added, the inference time per update will also increase as $O(t)$.

Looking back to Section 15.2.1, we see that constant time and space per filtering update can be achieved if the computation can be done recursively. Essentially, the filtering update in Equation (15.5) works by *summing out* the state variables of the previous time step to get the distribution for the new time step. Summing out variables is exactly what the **variable elimination** (Figure 14.11) algorithm does, and it turns out that running variable elimination with the variables in temporal order exactly mimics the operation of the recursive filtering update in Equation (15.5). The modified algorithm keeps at most two slices in memory at any one time: starting with slice 0, we add slice 1, then sum out slice 0, then add slice 2, then sum out slice 1, and so on. In this way, we can achieve constant space and time per filtering update. (The same performance can be achieved by suitable modifications to the clustering algorithm.) Exercise 15.17 asks you to verify this fact for the umbrella network.

So much for the good news; now for the bad news: It turns out that the "constant" for the per-update time and space complexity is, in almost all cases, exponential in the number of state variables. What happens is that, as the variable elimination proceeds, the factors grow to include all the state variables (or, more precisely, all those state variables that have parents in the previous time slice). The maximum factor size is $O(d^{n+k})$ and the total update cost per step is $O(nd^{n+k})$, where d is the domain size of the variables and k is the maximum number of parents of any state variable.

Of course, this is much less than the cost of HMM updating, which is $O(d^{2n})$, but it is still infeasible for large numbers of variables. This grim fact is somewhat hard to accept. What it means is that *even though we can use DBNs to* represent *very complex temporal processes with many sparsely connected variables, we cannot* reason *efficiently and exactly about those processes.* The DBN model itself, which represents the prior joint distribution over all the variables, is factorable into its constituent CPTs, but the posterior joint distribution conditioned on an observation sequence—that is, the forward message—is generally *not* factorable. So far, no one has found a way around this problem, despite the fact that many important areas of science and engineering would benefit enormously from its solution. Thus, we must fall back on approximate methods.

15.5.3 Approximate inference in DBNs

Section 14.5 described two approximation algorithms: likelihood weighting (Figure 14.15) and Markov chain Monte Carlo (MCMC, Figure 14.16). Of the two, the former is most easily adapted to the DBN context. (An MCMC filtering algorithm is described briefly in the notes at the end of the chapter.) We will see, however, that several improvements are required over the standard likelihood weighting algorithm before a practical method emerges.

Recall that likelihood weighting works by sampling the nonevidence nodes of the network in topological order, weighting each sample by the likelihood it accords to the observed evidence variables. As with the exact algorithms, we could apply likelihood weighting directly to an unrolled DBN, but this would suffer from the same problems of increasing time

and space requirements per update as the observation sequence grows. The problem is that the standard algorithm runs each sample in turn, all the way through the network. Instead, we can simply run all N samples together through the DBN, one slice at a time. The modified algorithm fits the general pattern of filtering algorithms, with the set of N samples as the forward message. The first key innovation, then, is to *use the samples themselves as an approximate representation of the current state distribution.* This meets the requirement of a "constant" time per update, although the constant depends on the number of samples required to maintain an accurate approximation. There is also no need to unroll the DBN, because we need to have in memory only the current slice and the next slice.

In our discussion of likelihood weighting in Chapter 14, we pointed out that the algorithm's accuracy suffers if the evidence variables are "downstream" from the variables being sampled, because in that case the samples are generated without any influence from the evidence. Looking at the typical structure of a DBN—say, the umbrella DBN in Figure 15.16—we see that indeed the early state variables will be sampled without the benefit of the later evidence. In fact, looking more carefully, we see that *none* of the state variables has *any* evidence variables among its ancestors! Hence, although the weight of each sample will depend on the evidence, the actual set of samples generated will be *completely independent* of the evidence. For example, even if the boss brings in the umbrella every day, the sampling process could still hallucinate endless days of sunshine. What this means in practice is that the fraction of samples that remain reasonably close to the actual series of events (and therefore have nonnegligible weights) drops exponentially with t, the length of the observation sequence. In other words, to maintain a given level of accuracy, we need to increase the number of samples exponentially with t. Given that a filtering algorithm that works in real time can use only a fixed number of samples, what happens in practice is that the error blows up after a very small number of update steps.

Clearly, we need a better solution. The second key innovation is to *focus the set of samples on the high-probability regions of the state space.* This can be done by throwing away samples that have very low weight, according to the observations, while replicating those that have high weight. In that way, the population of samples will stay reasonably close to reality. If we think of samples as a resource for modeling the posterior distribution, then it makes sense to use more samples in regions of the state space where the posterior is higher.

PARTICLE FILTERING

A family of algorithms called **particle filtering** is designed to do just that. Particle filtering works as follows: First, a population of N initial-state samples is created by sampling from the prior distribution $\mathbf{P}(\mathbf{X}_0)$. Then the update cycle is repeated for each time step:

1. Each sample is propagated forward by sampling the next state value $\mathbf{x}_{t+1}$ given the current value $\mathbf{x}_t$ for the sample, based on the transition model $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$.
2. Each sample is weighted by the likelihood it assigns to the new evidence, $P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1})$.
3. The population is *resampled* to generate a new population of N samples. Each new sample is selected from the current population; the probability that a particular sample is selected is proportional to its weight. The new samples are unweighted.

The algorithm is shown in detail in Figure 15.17, and its operation for the umbrella DBN is illustrated in Figure 15.18.

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 $\mathbf{P}(\mathbf{X}_0)$, transition model $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$, sensor model $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$
  persistent: S, a vector of samples of size N, initially generated from $\mathbf{P}(\mathbf{X}_0)$
  local variables: W, a vector of weights of size N

  for i = 1 to N **do**
    $S[i] \leftarrow$ sample from $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0 = S[i])$ /* step 1 */
    $W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ /* step 2 */
  $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W) /* step 3 */
  return S

Figure 15.17 The particle filtering algorithm implemented as a recursive update operation with state (the set of samples). Each of the sampling operations involves sampling the relevant slice variables in topological order, much as in PRIOR-SAMPLE. The WEIGHTED-SAMPLE-WITH-REPLACEMENT operation can be implemented to run in $O(N)$ expected time. The step numbers refer to the description in the text.

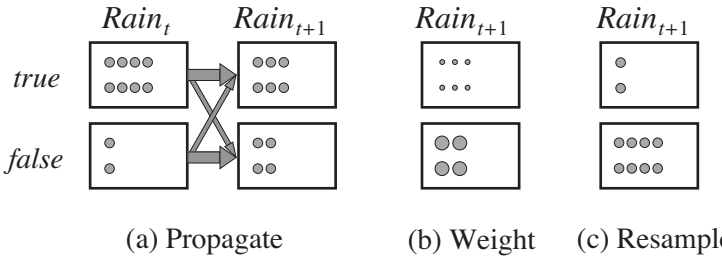


Figure 15.18 The particle filtering update cycle for the umbrella DBN with $N = 10$, showing the sample populations of each state. (a) At time t, 8 samples indicate $rain$ and 2 indicate $\neg rain$. Each is propagated forward by sampling the next state through the transition model. At time $t + 1$, 6 samples indicate $rain$ and 4 indicate $\neg rain$. (b) $\neg umbrella$ is observed at $t + 1$. Each sample is weighted by its likelihood for the observation, as indicated by the size of the circles. (c) A new set of 10 samples is generated by weighted random selection from the current set, resulting in 2 samples that indicate $rain$ and 8 that indicate $\neg rain$.

We can show that this algorithm is consistent—gives the correct probabilities as N tends to infinity—by considering what happens during one update cycle. We assume that the sample population starts with a correct representation of the forward message $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})$ at time t. Writing $N(\mathbf{x}_t \mid \mathbf{e}_{1:t})$ for the number of samples occupying state $\mathbf{x}_t$ after observations $\mathbf{e}_{1:t}$ have been processed, we therefore have

$$N(\mathbf{x}_t \mid \mathbf{e}_{1:t})/N = P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \tag{15.23}$$

for large N. Now we propagate each sample forward by sampling the state variables at $t + 1$, given the values for the sample at t. The number of samples reaching state $\mathbf{x}_{t+1}$ from each

$\mathbf{x}_t$ is the transition probability times the population of $\mathbf{x}_t$; hence, the total number of samples reaching $\mathbf{x}_{t+1}$ is

$$N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \ .$$

Now we weight each sample by its likelihood for the evidence at $t+1$. A sample in state $\mathbf{x}_{t+1}$ receives weight $P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1})$. The total weight of the samples in $\mathbf{x}_{t+1}$ after seeing $\mathbf{e}_{t+1}$ is therefore

$$W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) \ .$$

Now for the resampling step. Since each sample is replicated with probability proportional to its weight, the number of samples in state $\mathbf{x}_{t+1}$ after resampling is proportional to the total weight in $\mathbf{x}_{t+1}$ before resampling:

$$\begin{aligned}
N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})/N &= \alpha\, W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) \\
&= \alpha\, P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) \\
&= \alpha\, P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\
&= \alpha\, N P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \quad \text{(by 15.23)} \\
&= \alpha' P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \\
&= P(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) \quad \text{(by 15.5).}
\end{aligned}$$

Therefore the sample population after one update cycle correctly represents the forward message at time $t+1$.

Particle filtering is *consistent*, therefore, but is it *efficient*? In practice, it seems that the answer is yes: particle filtering seems to maintain a good approximation to the true posterior using a constant number of samples. Under certain assumptions—in particular, that the probabilities in the transition and sensor models are strictly greater than 0 and less than 1—it is possible to prove that the approximation maintains bounded error with high probability. On the practical side, the range of applications has grown to include many fields of science and engineering; some references are given at the end of the chapter.

15.6 Keeping Track of Many Objects

The preceding sections have considered—without mentioning it—state estimation problems involving a single object. In this section, we see what happens when two or more objects generate the observations. What makes this case different from plain old state estimation is that there is now the possibility of *uncertainty* about which object generated which observation. This is the **identity uncertainty** problem of Section 14.6.3 (page 544), now viewed in a temporal context. In the control theory literature, this is the **data association** problem—that is, the problem of associating observation data with the objects that generated them.

DATA ASSOCIATION

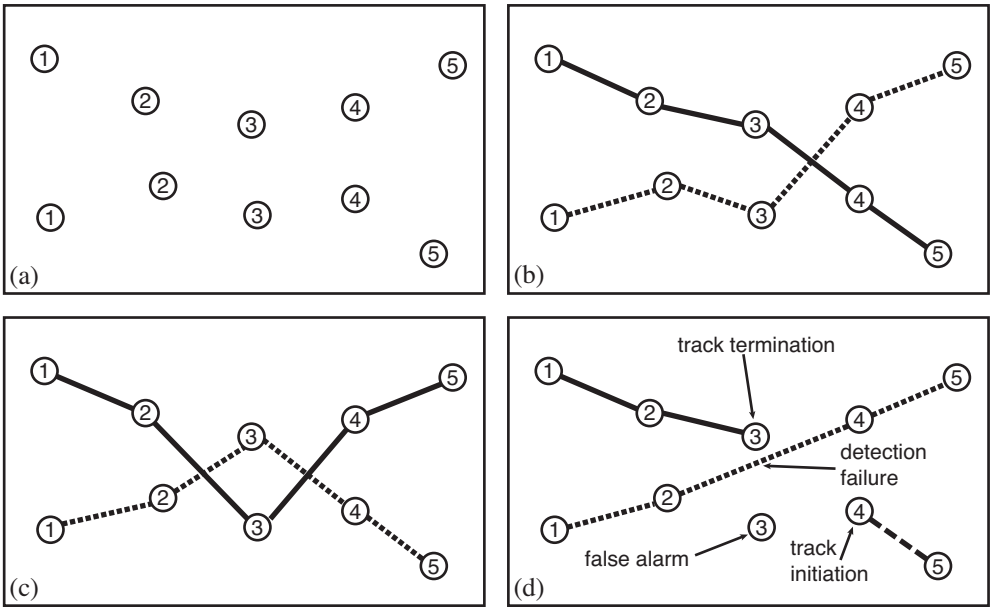


Figure 15.19 (a) Observations made of object locations in 2D space over five time steps. Each observation is labeled with the time step but does not identify the object that produced it. (b–c) Possible hypotheses about the underlying object tracks. (d) A hypothesis for the case in which false alarms, detection failures, and track initiation/termination are possible.

The data association problem was studied originally in the context of radar tracking, where reflected pulses are detected at fixed time intervals by a rotating radar antenna. At each time step, multiple blips may appear on the screen, but there is no direct observation of which blips at time t belong to which blips at time $t-1$. Figure 15.19(a) shows a simple example with two blips per time step for five steps. Let the two blip locations at time t be e_t^1 and e_t^2. (The labeling of blips within a time step as "1" and "2" is completely arbitrary and carries no information.) Let us assume, for the time being, that exactly two aircraft, A and B, generated the blips; their true positions are X_t^A and X_t^B. Just to keep things simple, we'll also assume that the each aircraft moves independently according to a known transition model—e.g., a linear Gaussian model as used in the Kalman filter (Section 15.4).

Suppose we try to write down the overall probability model for this scenario, just as we did for general temporal processes in Equation (15.3) on page 569. As usual, the joint distribution factors into contributions for each time step as follows:

$$P(x_{0:t}^A, x_{0:t}^B, e_{1:t}^1, e_{1:t}^2) = \\ P(x_0^A)P(x_0^B)\prod_{i=1}^{t} P(x_i^A \mid x_{i-1}^A)P(x_i^B \mid x_{i-1}^B)\,P(e_i^1, e_i^2 \mid x_i^A, x_i^B)\,. \tag{15.24}$$

We would like to factor the observation term $P(e_i^1, e_i^2 \mid x_i^A, x_i^B)$ into a product of two terms, one for each object, but this would require knowing which observation was generated by which object. Instead, we have to sum over all possible ways of associating the observations

with the objects. Some of those ways are shown in Figure 15.19(b–c); in general, for n objects and T time steps, there are $(n!)^T$ ways of doing it—an awfully large number.

Mathematically speaking, the "way of associating the observations with the objects" is a collection of unobserved random variable that identify the source of each observation. We'll write ω_t to denote the one-to-one mapping from objects to observations at time t, with $\omega_t(A)$ and $\omega_t(B)$ denoting the specific observations (1 or 2) that ω_t assigns to A and B. (For n objects, ω_t will have $n!$ possible values; here, $n! = 2$.) Because the labels "1" ad "2" on the observations are assigned arbitrarily, the prior on ω_t is uniform and ω_t is independent of the states of the objects, x_t^A and x_t^B). So we can condition the observation term $P(e_i^1, e_i^2 \mid x_i^A, x_i^B)$ on ω_t and then simplify:

$$\begin{aligned} P(e_i^1, e_i^2 \mid x_i^A, x_i^B) &= \sum_{\omega_i} P(e_i^1, e_i^2 \mid x_i^A, x_i^B, \omega_i) P(\omega_i \mid x_i^A, x_i^B) \\ &= \sum_{\omega_i} P(e_i^{\omega_i(A)} \mid x_i^A) P(e_i^{\omega_i(B)} \mid x_i^B) P(\omega_i \mid x_i^A, x_i^B) \\ &= \frac{1}{2} \sum_{\omega_i} P(e_i^{\omega_i(A)} \mid x_i^A) P(e_i^{\omega_i(B)} \mid x_i^B) \ . \end{aligned}$$

Plugging this into Equation (15.24), we get an expression that is only in terms of transition and sensor models for individual objects and observations.

As for all probability models, inference means summing out the variables other than the query and the evidence. For filtering in HMMs and DBNs, we were able to sum out the state variables from 1 to $t-1$ by a simple dynamic programming trick; for Kalman filters, we took advantage of special properties of Gaussians. For data association, we are less fortunate. There is no (known) efficient exact algorithm, for the same reason that there is none for the switching Kalman filter (page 589): the filtering distribution $P(x_t^A \mid e_{1:t}^1, e_{1:t}^2)$ for object A ends up as a mixture of exponentially many distributions, one for each way of picking a sequence of observations to assign to A.

As a result of the complexity of exact inference, many different approximate methods have been used. The simplest approach is to choose a single "best" assignment at each time step, given the predicted positions of the objects at the current time step. This assignment associates observations with objects and enables the track of each object to be updated and a prediction made for the next time step. For choosing the "best" assignment, it is common to use the so-called **nearest-neighbor filter**, which repeatedly chooses the closest pairing of predicted position and observation and adds that pairing to the assignment. The nearest-neighbor filter works well when the objects are well separated in state space and the prediction uncertainty and observation error are small—in other words, when there is no possibility of confusion. When there is more uncertainty as to the correct assignment, a better approach is to choose the assignment that maximizes the joint probability of the current observations given the predicted positions. This can be done very efficiently using the **Hungarian algorithm** (Kuhn, 1955), even though there are $n!$ assignments to choose from.

NEAREST-NEIGHBOR FILTER

HUNGARIAN ALGORITHM

Any method that commits to a single best assignment at each time step fails miserably under more difficult conditions. In particular, if the algorithm commits to an incorrect assignment, the prediction at the next time step may be significantly wrong, leading to more

(a) (b)

Figure 15.20 Images from (a) upstream and (b) downstream surveillance cameras roughly two miles apart on Highway 99 in Sacramento, California. The boxed vehicle has been identified at both cameras.

incorrect assignments, and so on. Two modern approaches turn out to be much more effective. A **particle filtering** algorithm (see page 598) for data association works by maintaining a large collection of possible current assignments. An **MCMC** algorithm explores the space of assignment histories—for example, Figure 15.19(b–c) might be states in the MCMC state space—and can change its mind about previous assignment decisions. Current MCMC data association methods can handle many hundreds of objects in real time while giving a good approximation to the true posterior distributions.

The scenario described so far involved n known objects generating n observations at each time step. Real application of data association are typically much more complicated.

FALSE ALARM
CLUTTER
DETECTION FAILURE

Often, the reported observations include **false alarms** (also known as **clutter**), which are not caused by real objects. **Detection failures** can occur, meaning that no observation is reported for a real object. Finally, new objects arrive and old ones disappear. These phenomena, which create even more possible worlds to worry about, are illustrated in Figure 15.19(d).

Figure 15.20 shows two images from widely separated cameras on a California freeway. In this application, we are interested in two goals: estimating the time it takes, under current traffic conditions, to go from one place to another in the freeway system; and measuring *demand*, i.e., how many vehicles travel between any two points in the system at particular times of the day and on particular days of the week. Both goals require solving the data association problem over a wide area with many cameras and tens of thousands of vehicles per hour. With visual surveillance, false alarms are caused by moving shadows, articulated vehicles, reflections in puddles, etc.; detection failures are caused by occlusion, fog, darkness, and lack of visual contrast; and vehicles are constantly entering and leaving the freeway system. Furthermore, the appearance of any given vehicle can change dramatically between cameras depending on lighting conditions and vehicle pose in the image, and the transition model changes as traffic jams come and go. Despite these problems, modern data association algorithms have been successful in estimating traffic parameters in real-world settings.

Data association is an essential foundation for keeping track of a complex world, because without it there is no way to combine multiple observations of any given object. When objects in the world interact with each other in complex activities, understanding the world requires combining data association with the relational and open-universe probability models of Section 14.6.3. This is currently an active area of research.

15.7 SUMMARY

This chapter has addressed the general problem of representing and reasoning about probabilistic temporal processes. The main points are as follows:

- The changing state of the world is handled by using a set of random variables to represent the state at each point in time.
- Representations can be designed to satisfy the **Markov property**, so that the future is independent of the past given the present. Combined with the assumption that the process is **stationary**—that is, the dynamics do not change over time—this greatly simplifies the representation.
- A temporal probability model can be thought of as containing a **transition model** describing the state evolution and a **sensor model** describing the observation process.
- The principal inference tasks in temporal models are **filtering**, **prediction**, **smoothing**, and computing the **most likely explanation**. Each of these can be achieved using simple, recursive algorithms whose run time is linear in the length of the sequence.
- Three families of temporal models were studied in more depth: **hidden Markov models**, **Kalman filters**, and **dynamic Bayesian networks** (which include the other two as special cases).
- Unless special assumptions are made, as in Kalman filters, exact inference with many state variables is intractable. In practice, the **particle filtering** algorithm seems to be an effective approximation algorithm.
- When trying to keep track of many objects, uncertainty arises as to which observations belong to which objects—the **data association** problem. The number of association hypotheses is typically intractably large, but MCMC and particle filtering algorithms for data association work well in practice.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

Many of the basic ideas for estimating the state of dynamical systems came from the mathematician C. F. Gauss (1809), who formulated a deterministic least-squares algorithm for the problem of estimating orbits from astronomical observations. A. A. Markov (1913) developed what was later called the **Markov assumption** in his analysis of stochastic processes;

he estimated a first-order Markov chain on letters from the text of *Eugene Onegin*. The general theory of Markov chains and their mixing times is covered by Levin *et al.* (2008).

Significant classified work on filtering was done during World War II by Wiener (1942) for continuous-time processes and by Kolmogorov (1941) for discrete-time processes. Although this work led to important technological developments over the next 20 years, its use of a frequency-domain representation made many calculations quite cumbersome. Direct state-space modeling of the stochastic process turned out to be simpler, as shown by Peter Swerling (1959) and Rudolf Kalman (1960). The latter paper described what is now known as the Kalman filter for forward inference in linear systems with Gaussian noise; Kalman's results had, however, been obtained previously by the Danish statistician Thorvold Thiele (1880) and by the Russian mathematician Ruslan Stratonovich (1959), whom Kalman met in Moscow in 1960. After a visit to NASA Ames Research Center in 1960, Kalman saw the applicability of the method to the tracking of rocket trajectories, and the filter was later implemented for the Apollo missions. Important results on smoothing were derived by Rauch *et al.* (1965), and the impressively named Rauch–Tung–Striebel smoother is still a standard technique today. Many early results are gathered in Gelb (1974). Bar-Shalom and Fortmann (1988) give a more modern treatment with a Bayesian flavor, as well as many references to the vast literature on the subject. Chatfield (1989) and Box *et al.* (1994) cover the control theory approach to time series analysis.

The hidden Markov model and associated algorithms for inference and learning, including the forward–backward algorithm, were developed by Baum and Petrie (1966). The Viterbi algorithm first appeared in (Viterbi, 1967). Similar ideas also appeared independently in the Kalman filtering community (Rauch *et al.*, 1965). The forward–backward algorithm was one of the main precursors of the general formulation of the EM algorithm (Dempster *et al.*, 1977); see also Chapter 20. Constant-space smoothing appears in Binder *et al.* (1997b), as does the divide-and-conquer algorithm developed in Exercise 15.3. Constant-time fixed-lag smoothing for HMMs first appeared in Russell and Norvig (2003). HMMs have found many applications in language processing (Charniak, 1993), speech recognition (Rabiner and Juang, 1993), machine translation (Och and Ney, 2003), computational biology (Krogh *et al.*, 1994; Baldi *et al.*, 1994), financial economics Bhar and Hamori (2004) and other fields. There have been several extensions to the basic HMM model, for example the Hierarchical HMM (Fine *et al.*, 1998) and Layered HMM (Oliver *et al.*, 2004) introduce structure back into the model, replacing the single state variable of HMMs.

Dynamic Bayesian networks (DBNs) can be viewed as a sparse encoding of a Markov process and were first used in AI by Dean and Kanazawa (1989b), Nicholson and Brady (1992), and Kjaerulff (1992). The last work extends the HUGIN Bayes net system to accommodate dynamic Bayesian networks. The book by Dean and Wellman (1991) helped popularize DBNs and the probabilistic approach to planning and control within AI. Murphy (2002) provides a thorough analysis of DBNs.

Dynamic Bayesian networks have become popular for modeling a variety of complex motion processes in computer vision (Huang *et al.*, 1994; Intille and Bobick, 1999). Like HMMs, they have found applications in speech recognition (Zweig and Russell, 1998; Richardson *et al.*, 2000; Stephenson *et al.*, 2000; Nefian *et al.*, 2002; Livescu *et al.*, 2003), ge-

nomics (Murphy and Mian, 1999; Perrin *et al.*, 2003; Husmeier, 2003) and robot localization (Theocharous *et al.*, 2004). The link between HMMs and DBNs, and between the forward–backward algorithm and Bayesian network propagation, was made explicitly by Smyth *et al.* (1997). A further unification with Kalman filters (and other statistical models) appears in Roweis and Ghahramani (1999). Procedures exist for learning the parameters (Binder *et al.*, 1997a; Ghahramani, 1998) and structures (Friedman *et al.*, 1998) of DBNs.

The particle filtering algorithm described in Section 15.5 has a particularly interesting history. The first sampling algorithms for particle filtering (also called sequential Monte Carlo methods) were developed in the control theory community by Handschin and Mayne (1969), and the resampling idea that is the core of particle filtering appeared in a Russian control journal (Zaritskii *et al.*, 1975). It was later reinvented in statistics as **sequential importance-sampling resampling**, or **SIR** (Rubin, 1988; Liu and Chen, 1998), in control theory as particle filtering (Gordon *et al.*, 1993; Gordon, 1994), in AI as **survival of the fittest** (Kanazawa *et al.*, 1995), and in computer vision as **condensation** (Isard and Blake, 1996). The paper by Kanazawa *et al.* (1995) includes an improvement called **evidence reversal** whereby the state at time $t+1$ is sampled conditional on both the state at time t *and the evidence at time* $t+1$. This allows the evidence to influence sample generation directly and was proved by Doucet (1997) and Liu and Chen (1998) to reduce the approximation error. Particle filtering has been applied in many areas, including tracking complex motion patterns in video (Isard and Blake, 1996), predicting the stock market (de Freitas *et al.*, 2000), and diagnosing faults on planetary rovers (Verma *et al.*, 2004). A variant called the **Rao-Blackwellized particle filter** or RBPF (Doucet *et al.*, 2000; Murphy and Russell, 2001) applies particle filtering to a subset of state variables and, for each particle, performs exact inference on the remaining variables conditioned on the value sequence in the particle. In some cases RBPF works well with thousands of state variables. An application of RBPF to localization and mapping in robotics is described in Chapter 25. The book by Doucet *et al.* (2001) collects many important papers on **sequential Monte Carlo** (SMC) algorithms, of which particle filtering is the most important instance. Pierre Del Moral and colleagues have performed extensive theoretical analyses of SMC algorithms (Del Moral, 2004; Del Moral *et al.*, 2006).

EVIDENCE REVERSAL

RAO-BLACKWELLIZED PARTICLE FILTER

SEQUENTIAL MONTE CARLO

MCMC methods (see Section 14.5.2) can be applied to the filtering problem; for example, Gibbs sampling can be applied directly to an unrolled DBN. To avoid the problem of increasing update times as the unrolled network grows, the **decayed MCMC** filter (Marthi *et al.*, 2002) prefers to sample more recent state variables, with a probability that decays as $1/k^2$ for a variable k steps into the past. Decayed MCMC is a provably nondivergent filter. Nondivergence theorems can also be obtained for certain types of **assumed-density filter**. An assumed-density filter assumes that the posterior distribution over states at time t belongs to a particular finitely parameterized family; if the projection and update steps take it outside this family, the distribution is projected back to give the best approximation within the family. For DBNs, the Boyen–Koller algorithm (Boyen *et al.*, 1999) and the **factored frontier** algorithm (Murphy and Weiss, 2001) assume that the posterior distribution can be approximated well by a product of small factors. Variational techniques (see Chapter 14) have also been developed for temporal models. Ghahramani and Jordan (1997) discuss an approximation algorithm for the **factorial HMM**, a DBN in which two or more independently evolving

DECAYED MCMC

ASSUMED-DENSITY FILTER

FACTORED FRONTIER

FACTORIAL HMM

Markov chains are linked by a shared observation stream. Jordan *et al.* (1998) cover a number of other applications.

Data association for multitarget tracking was first described in a probabilistic setting by Sittler (1964). The first practical algorithm for large-scale problems was the "multiple hypothesis tracker" or MHT algorithm (Reid, 1979). Many important papers are collected by Bar-Shalom and Fortmann (1988) and Bar-Shalom (1992). The development of an MCMC algorithm for data association is due to Pasula *et al.* (1999), who applied it to traffic surveillance problems. Oh *et al.* (2009) provide a formal analysis and extensive experimental comparisons to other methods. Schulz *et al.* (2003) describe a data association method based on particle filtering. Ingemar Cox analyzed the complexity of data association (Cox, 1993; Cox and Hingorani, 1994) and brought the topic to the attention of the vision community. He also noted the applicability of the polynomial-time Hungarian algorithm to the problem of finding most-likely assignments, which had long been considered an intractable problem in the tracking community. The algorithm itself was published by Kuhn (1955), based on translations of papers published in 1931 by two Hungarian mathematicians, Dénes König and Jenö Egerváry. The basic theorem had been derived previously, however, in an unpublished Latin manuscript by the famous Prussian mathematician Carl Gustav Jacobi (1804–1851).

EXERCISES

15.1 Show that any second-order Markov process can be rewritten as a first-order Markov process with an augmented set of state variables. Can this always be done *parsimoniously*, i.e., without increasing the number of parameters needed to specify the transition model?

15.2 In this exercise, we examine what happens to the probabilities in the umbrella world in the limit of long time sequences.

a. Suppose we observe an unending sequence of days on which the umbrella appears. Show that, as the days go by, the probability of rain on the current day increases monotonically toward a fixed point. Calculate this fixed point.

b. Now consider *forecasting* further and further into the future, given just the first two umbrella observations. First, compute the probability $P(r_{2+k}|u_1, u_2)$ for $k = 1 \ldots 20$ and plot the results. You should see that the probability converges towards a fixed point. Prove that the exact value of this fixed point is 0.5.

15.3 This exercise develops a space-efficient variant of the forward–backward algorithm described in Figure 15.4 (page 576). We wish to compute $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $k = 1, \ldots, t$. This will be done with a divide-and-conquer approach.

a. Suppose, for simplicity, that t is odd, and let the halfway point be $h = (t + 1)/2$. Show that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed for $k = 1, \ldots, h$ given just the initial forward message $\mathbf{f}_{1:0}$, the backward message $\mathbf{b}_{h+1:t}$, and the evidence $\mathbf{e}_{1:h}$.

b. Show a similar result for the second half of the sequence.

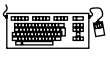
c. Given the results of (a) and (b), a recursive divide-and-conquer algorithm can be constructed by first running forward along the sequence and then backward from the end, storing just the required messages at the middle and the ends. Then the algorithm is called on each half. Write out the algorithm in detail.

d. Compute the time and space complexity of the algorithm as a function of t, the length of the sequence. How does this change if we divide the input into more than two pieces?

15.4 On page 577, we outlined a flawed procedure for finding the most likely state sequence, given an observation sequence. The procedure involves finding the most likely state at each time step, using smoothing, and returning the sequence composed of these states. Show that, for some temporal probability models and observation sequences, this procedure returns an impossible state sequence (i.e., the posterior probability of the sequence is zero).

15.5 Equation (15.12) describes the filtering process for the matrix formulation of HMMs. Give a similar equation for the calculation of likelihoods, which was described generically in Equation (15.7).

15.6 Consider the vacuum worlds of Figure 4.18 (perfect sensing) and Figure 15.7 (noisy sensing). Suppose that the robot receives an observation sequence such that, with perfect sensing, there is exactly one possible location it could be in. Is this location necessarily the most probable location under noisy sensing for sufficiently small noise probability ϵ? Prove your claim or find a counterexample.



15.7 In Section 15.3.2, the prior distribution over locations is uniform and the transition model assumes an equal probability of moving to any neighboring square. What if those assumptions are wrong? Suppose that the initial location is actually chosen uniformly from the northwest quadrant of the room and the *Move* action actually tends to move southeast. Keeping the HMM model fixed, explore the effect on localization and path accuracy as the southeasterly tendency increases, for different values of ϵ.

15.8 Consider a version of the vacuum robot (page 582) that has the policy of going straight for as long as it can; only when it encounters an obstacle does it change to a new (randomly selected) heading. To model this robot, each state in the model consists of a *(location, heading)* pair. Implement this model and see how well the Viterbi algorithm can track a robot with this model. The robot's policy is more constrained than the random-walk robot; does that mean that predictions of the most likely path are more accurate?

15.9 This exercise is concerned with filtering in an environment with no landmarks. Consider a vacuum robot in an empty room, represented by an $n \times m$ rectangular grid. The robot's location is hidden; the only evidence available to the observer is a noisy location sensor that gives an approximation to the robot's location. If the robot is at location (x, y) then with probability .1 the sensor gives the correct location, with probability .05 each it reports one of the 8 locations immediately surrounding (x, y), with probability .025 each it reports one of the 16 locations that surround those 8, and with the remaining probability of .1 it reports "no reading." The robot's policy is to pick a direction and follow it with probability .8 on each step; the robot switches to a randomly selected new heading with probability .2 (or with

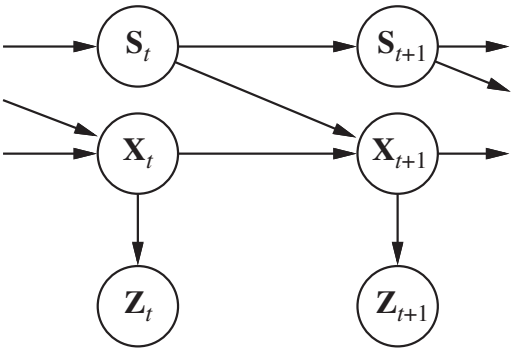


Figure 15.21 A Bayesian network representation of a switching Kalman filter. The switching variable S_t is a discrete state variable whose value determines the transition model for the continuous state variables $\mathbf{X}_t$. For any discrete state i, the transition model $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t, S_t = i)$ is a linear Gaussian model, just as in a regular Kalman filter. The transition model for the discrete state, $\mathbf{P}(S_{t+1}|S_t)$, can be thought of as a matrix, as in a hidden Markov model.

probability 1 if it encounters a wall). Implement this as an HMM and do filtering to track the robot. How accurately can we track the robot's path?

15.10 Often, we wish to monitor a continuous-state system whose behavior switches unpredictably among a set of k distinct "modes." For example, an aircraft trying to evade a missile can execute a series of distinct maneuvers that the missile may attempt to track. A Bayesian network representation of such a **switching Kalman filter** model is shown in Figure 15.21.

a. Suppose that the discrete state S_t has k possible values and that the prior continuous state estimate $\mathbf{P}(\mathbf{X}_0)$ is a multivariate Gaussian distribution. Show that the prediction $\mathbf{P}(\mathbf{X}_1)$ is a **mixture of Gaussians**—that is, a weighted sum of Gaussians such that the weights sum to 1.

b. Show that if the current continuous state estimate $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ is a mixture of m Gaussians, then in the general case the updated state estimate $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1})$ will be a mixture of km Gaussians.

c. What aspect of the temporal process do the weights in the Gaussian mixture represent?

The results in (a) and (b) show that the representation of the posterior grows without limit even for switching Kalman filters, which are among the simplest hybrid dynamic models.

15.11 Complete the missing step in the derivation of Equation (15.19) on page 586, the first update step for the one-dimensional Kalman filter.

15.12 Let us examine the behavior of the variance update in Equation (15.20) (page 587).

a. Plot the value of σ_t^2 as a function of t, given various values for σ_x^2 and σ_z^2.

b. Show that the update has a fixed point σ^2 such that $\sigma_t^2 \to \sigma^2$ as $t \to \infty$, and calculate the value of σ^2.

c. Give a qualitative explanation for what happens as $\sigma_x^2 \to 0$ and as $\sigma_z^2 \to 0$.

15.13 A professor wants to know if students are getting enough sleep. Each day, the professor observes whether the students sleep in class, and whether they have red eyes. The professor has the following domain theory:

- The prior probability of getting enough sleep, with no observations, is 0.7.
- The probability of getting enough sleep on night t is 0.8 given that the student got enough sleep the previous night, and 0.3 if not.
- The probability of having red eyes is 0.2 if the student got enough sleep, and 0.7 if not.
- The probability of sleeping in class is 0.1 if the student got enough sleep, and 0.3 if not.

Formulate this information as a dynamic Bayesian network that the professor could use to filter or predict from a sequence of observations. Then reformulate it as a hidden Markov model that has only a single observation variable. Give the complete probability tables for the model.

15.14 For the DBN specified in Exercise 15.13 and for the evidence values

$\mathbf{e}_1 =$ not red eyes, not sleeping in class
$\mathbf{e}_2 =$ red eyes, not sleeping in class
$\mathbf{e}_3 =$ red eyes, sleeping in class

perform the following computations:

a. State estimation: Compute $P(EnoughSleep_t|\mathbf{e}_{1:t})$ for each of $t = 1, 2, 3$.
b. Smoothing: Compute $P(EnoughSleep_t|\mathbf{e}_{1:3})$ for each of $t = 1, 2, 3$.
c. Compare the filtered and smoothed probabilities for $t = 1$ and $t = 2$.

15.15 Suppose that a particular student shows up with red eyes and sleeps in class every day. Given the model described in Exercise 15.13, explain why the probability that the student had enough sleep the previous night converges to a fixed point rather than continuing to go down as we gather more days of evidence. What is the fixed point? Answer this both numerically (by computation) and analytically.

15.16 This exercise analyzes in more detail the persistent-failure model for the battery sensor in Figure 15.15(a) (page 594).

a. Figure 15.15(b) stops at $t = 32$. Describe qualitatively what should happen as $t \to \infty$ if the sensor continues to read 0.
b. Suppose that the external temperature affects the battery sensor in such a way that transient failures become more likely as temperature increases. Show how to augment the DBN structure in Figure 15.15(a), and explain any required changes to the CPTs.
c. Given the new network structure, can battery readings be used by the robot to infer the current temperature?

15.17 Consider applying the variable elimination algorithm to the umbrella DBN unrolled for three slices, where the query is $\mathbf{P}(R_3|u_1, u_2, u_3)$. Show that the space complexity of the algorithm—the size of the largest factor—is the same, regardless of whether the rain variables are eliminated in forward or backward order.

16 MAKING SIMPLE DECISIONS

In which we see how an agent should make decisions so that it gets what it wants—on average, at least.

In this chapter, we fill in the details of how utility theory combines with probability theory to yield a decision-theoretic agent—an agent that can make rational decisions based on what it believes and what it wants. Such an agent can make decisions in contexts in which uncertainty and conflicting goals leave a logical agent with no way to decide: a goal-based agent has a binary distinction between good (goal) and bad (non-goal) states, while a decision-theoretic agent has a continuous measure of outcome quality.

Section 16.1 introduces the basic principle of decision theory: the maximization of expected utility. Section 16.2 shows that the behavior of any rational agent can be captured by supposing a utility function that is being maximized. Section 16.3 discusses the nature of utility functions in more detail, and in particular their relation to individual quantities such as money. Section 16.4 shows how to handle utility functions that depend on several quantities. In Section 16.5, we describe the implementation of decision-making systems. In particular, we introduce a formalism called a **decision network** (also known as an **influence diagram**) that extends Bayesian networks by incorporating actions and utilities. The remainder of the chapter discusses issues that arise in applications of decision theory to expert systems.

16.1 COMBINING BELIEFS AND DESIRES UNDER UNCERTAINTY

Decision theory, in its simplest form, deals with choosing among actions based on the desirability of their *immediate* outcomes; that is, the environment is assumed to be episodic in the sense defined on page 43. (This assumption is relaxed in Chapter 17.) In Chapter 3 we used the notation $\text{RESULT}(s_0, a)$ for the state that is the deterministic outcome of taking action a in state s_0. In this chapter we deal with nondeterministic partially observable environments. Since the agent may not know the current state, we omit it and define $\text{RESULT}(a)$ as a *random variable* whose values are the possible outcome states. The probability of outcome s', given evidence observations $\mathbf{e}$, is written

$$P(\text{RESULT}(a) = s' \mid a, \mathbf{e}) \ ,$$