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

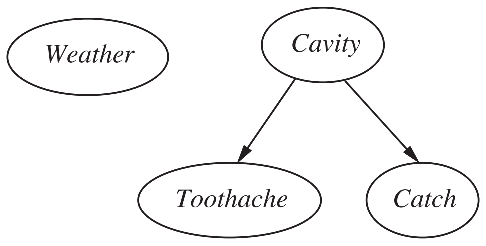


Figure 14.1 A simple Bayesian network in which $Weather$ is independent of the other three variables and $Toothache$ and $Catch$ are conditionally independent, given $Cavity$.

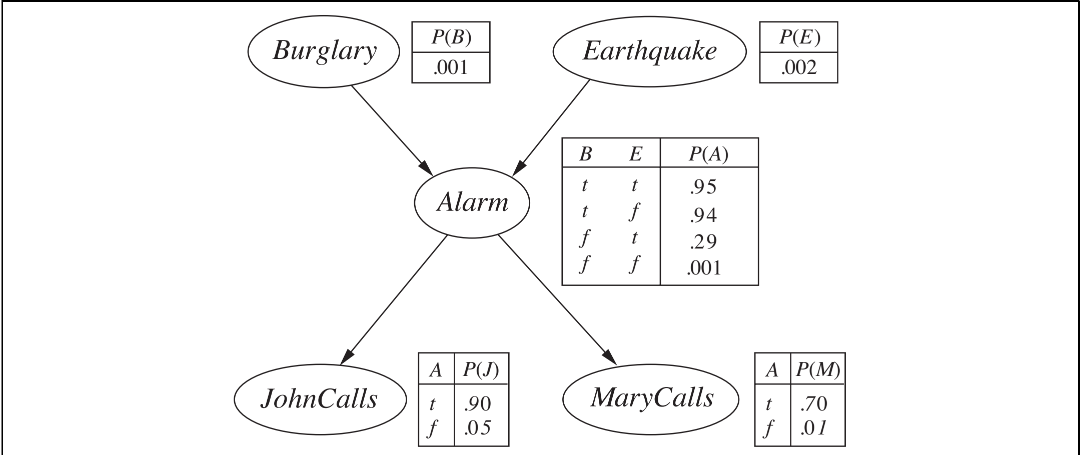


Figure 14.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for $Burglary$, $Earthquake$, $Alarm$, $JohnCalls$, and $MaryCalls$, respectively.

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 $\mathbf{P}(X_i \,|\, Parents(X_i))$; this is a true statement, but until we assign semantics to the network as a whole, we should think of them just as numbers $\theta(X_i \,|\, Parents(X_i))$.

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

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

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 $\theta(X_i \,|\, Parents(X_i))$ are exactly the conditional probabilities $\mathbf{P}(X_i \,|\, Parents(X_i))$ implied by the joint distribution (see Exercise 14.2). Hence, we can rewrite Equation (14.1) as

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

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

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.

LOCALLY STRUCTURED

SPARSE

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.

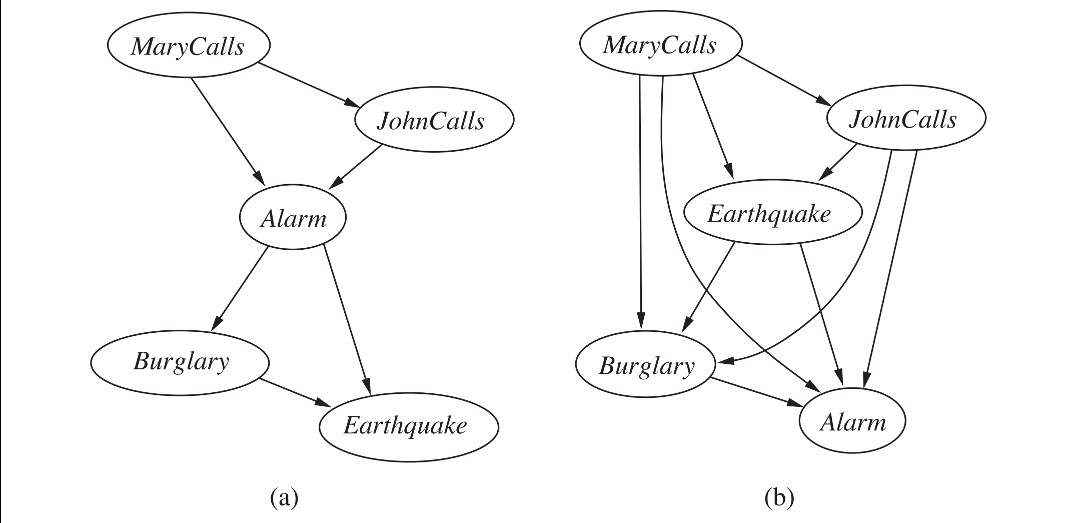


Figure 14.3 Network structure depends on order of introduction. In each network, we have introduced nodes in top-to-bottom order.

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 assessing 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.

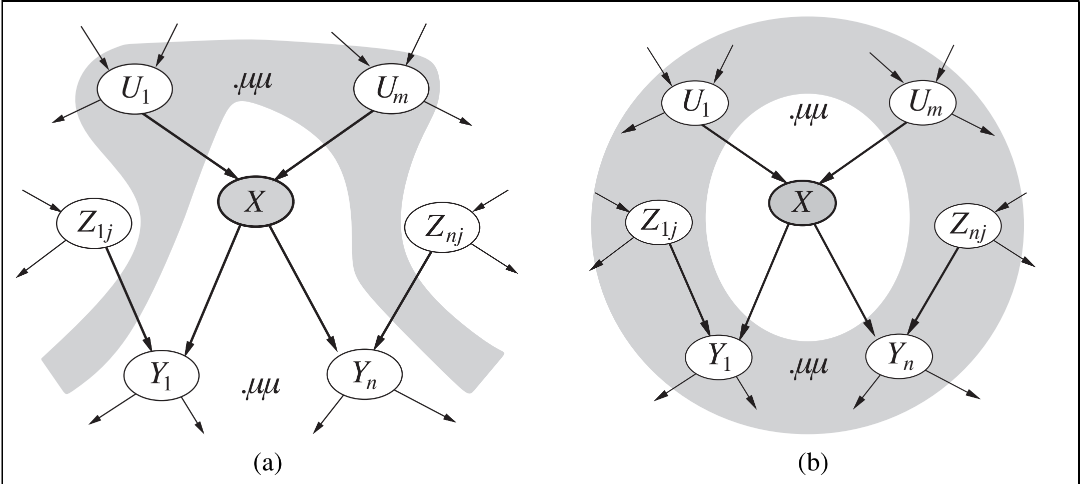


Figure 14.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the gray area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the gray area).

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

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

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

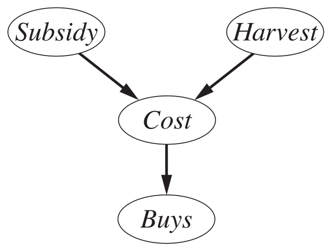


Figure 14.5 A simple network with discrete variables ($Subsidy$ and $Buys$) and continuous variables ($Harvest$ and $Cost$).

possible values into a fixed set of intervals. For example, temperatures could be divided into (<0°C), (0°C−100°C), and (>100°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

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

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)

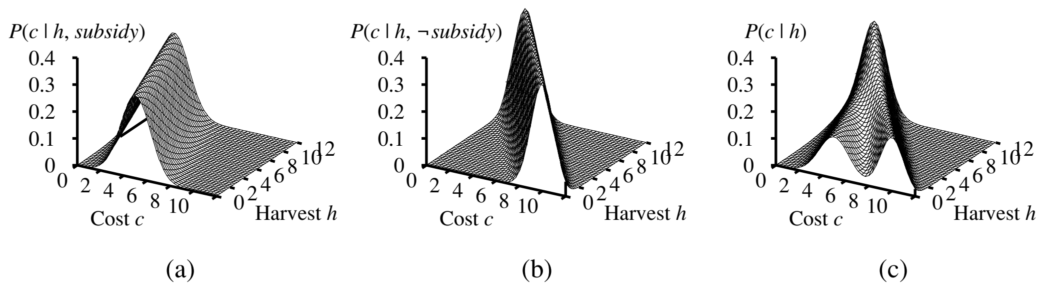


Figure 14.6 The graphs in (a) and (b) show the probability distribution over $Cost$ as a function of $Harvest$ size, with $Subsidy$ true and false, respectively. Graph (c) shows the distribution $P(Cost \mid Harvest)$, obtained by summing over the two subsidy cases.

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.

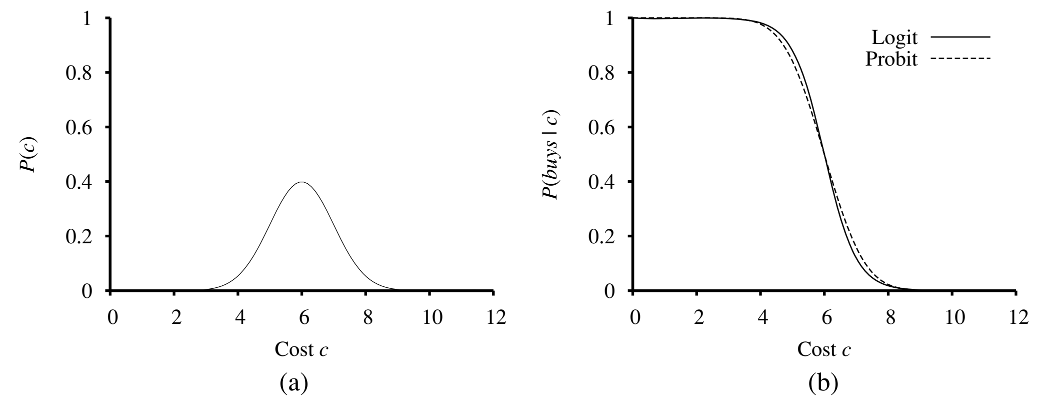


Figure 14.7 (a) A normal (Gaussian) distribution for the cost threshold, centered on $\mu = 6.0$ with standard deviation $\sigma = 1.0$. (b) Logit and probit distributions for the probability of $buys$ given $cost$, for the parameters $\mu = 6.0$ and $\sigma = 1.0$.

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

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

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

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

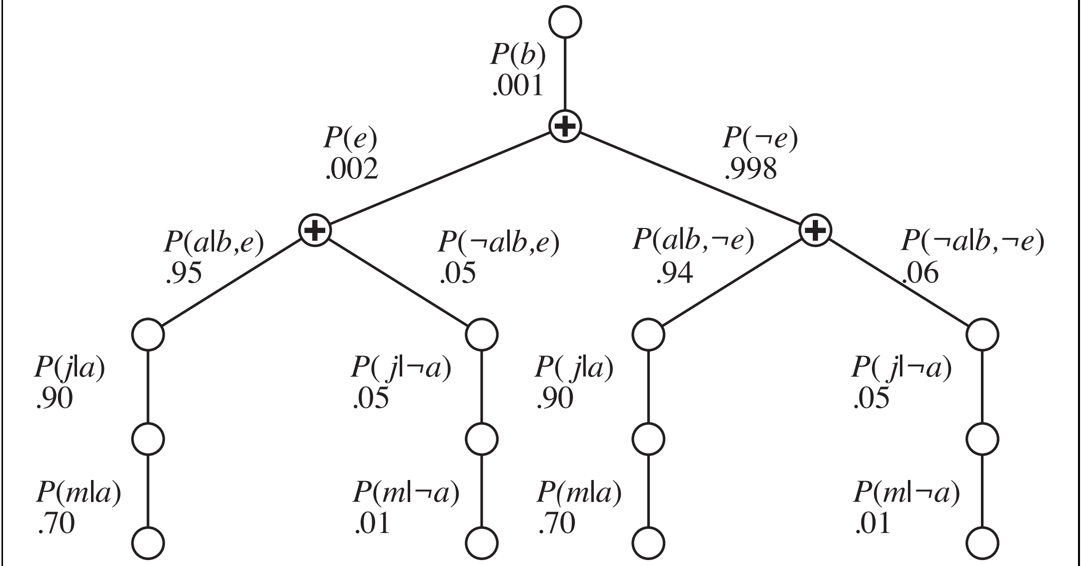


Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

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

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

```
function ENUMERATE-ALL($vars$, e) returns a real number
  if EMPTY?($vars$) then return 1.0
  $Y$ ← FIRST($vars$)
  if $Y$ has value $y$ in e
     then return $P(y \mid parents(Y))$ × ENUMERATE-ALL(REST($vars$), e)
     else return $\sum_y P(y \mid parents(Y))$ × 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.

POINTWISE PRODUCT

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

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 $\mathbf{P}(X_1, \ldots, X_n)$

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

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

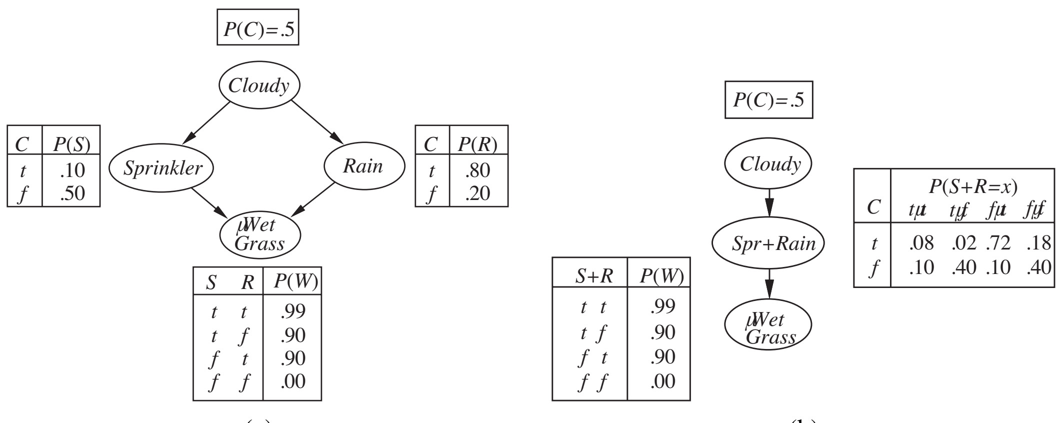
SINGLY CONNECTED

POLYTREE

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.

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*



C	P(S)
t	.10
f	.50

C	P(R)
t	.80
f	.20

S	R	P(W)
t	t	.99
t	f	.90
f	t	.90
f	f	.00

S+R	P(W)
t t	.99
t f	.90
f t	.90
f f	.00

	P(S+R=x)			
C	tt	tf	ft	ff
t	.08	.02	.72	.18
f	.10	.40	.10	.40

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

MONTE CARLO

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.

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 $\mathbf{P}(X_1, \ldots, X_n)$

  x ← an event with $n$ elements
  foreach variable $X_i$ in $X_1, \ldots, X_n$ do
      x[$i$] ← 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:

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

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

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

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 **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 **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 P($X$|e)
  inputs: $X$, the query variable
          e, observed values for variables E
          $bn$, a Bayesian network
          $N$, the total number of samples to be generated
  local variables: N, a vector of counts for each value of $X$, initially zero

  for $j$ = 1 to $N$ do
     x ← PRIOR-SAMPLE($bn$)
     if x is consistent with e then
        N[$x$] ← 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 $\mathbf{x}[i] \leftarrow$ 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 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($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: $\mathbf{N}$, a vector of counts for each value of $X$, initially zero
                   $\mathbf{Z}$, the nonevidence variables in $bn$
                   $\mathbf{x}$, the current state of the network, initially copied from $\mathbf{e}$

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

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 $P(\mathbf{x} \mid \mathbf{e})$, (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

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

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

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

We can think of the loop "**for each** Z_i in $\mathbf{Z}$ **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 $P(\mathbf{x} \mid \mathbf{e})$ 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 $\mathbf{P}(X_i \mid \overline{\mathbf{x}_i}, \mathbf{e})$—in a Bayesian network. Recall from page 517 that a variable is independent of all other variables given its Markov blanket; hence,

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

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:

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

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.

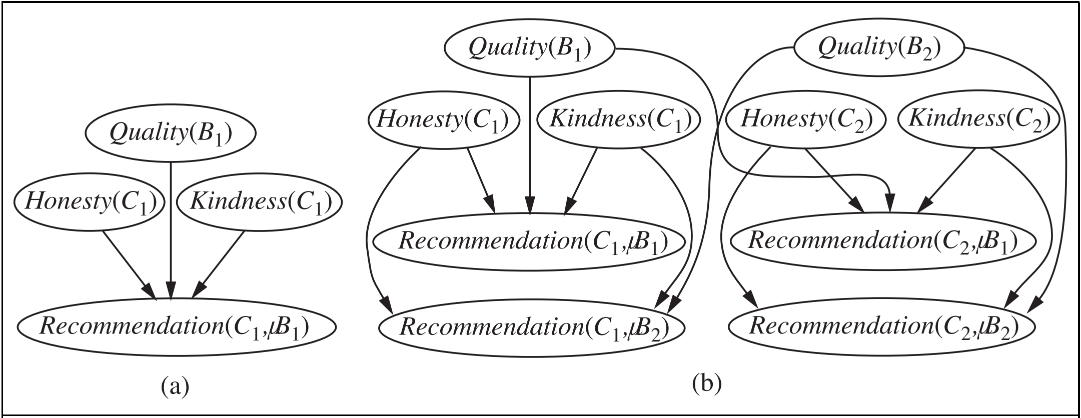


Figure 14.17 (a) Bayes net for a single customer C_1 recommending a single book B_1. $Honest(C_1)$ is Boolean, while the other variables have integer values from 1 to 5. (b) Bayes net with two customers and two books.

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

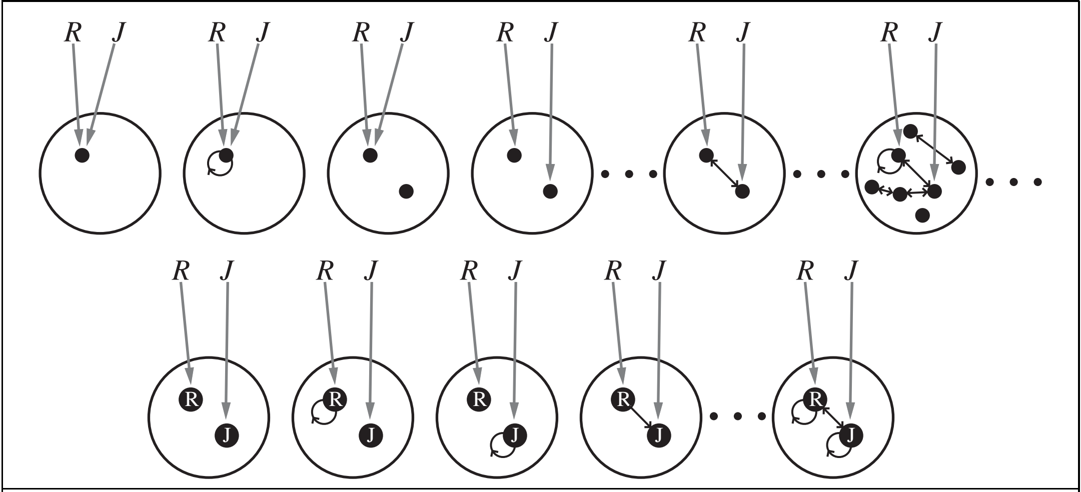


Figure 14.18 Top: Some members of the set of all possible worlds for a language with two constant symbols, R and J, and one binary relation symbol, under the standard semantics for first-order logic. Bottom: the possible worlds under database semantics. The interpretation of the constant symbols is fixed, 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 $\mathit{Customer}$ and Book, and the type signatures for the functions and predicates are as follows:

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

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: $\mathit{Honest}(C_1),\ \mathit{Quality}(B_2),\ \mathit{Recommendation}(C_1, B_2)$, 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):

$$\begin{aligned}&\mathit{Honest}(c) \sim \langle 0.99, 0.01\rangle\\&\mathit{Kindness}(c) \sim \langle 0.1, 0.1, 0.2, 0.3, 0.3\rangle\\&\mathit{Quality}(b) \sim \langle 0.05, 0.2, 0.4, 0.2, 0.15\rangle\\&\mathit{Recommendation}(c, b) \sim \mathit{RecCPT}(\mathit{Honest}(c), \mathit{Kindness}(c), \mathit{Quality}(b))\end{aligned}$$

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, $\mathit{Recommendation}(c, b)$ is independent of $\mathit{Kindness}(c)$ and $\mathit{Quality}(b)$ when $\mathit{Honest}(c) = \mathit{false}$:

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

[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.

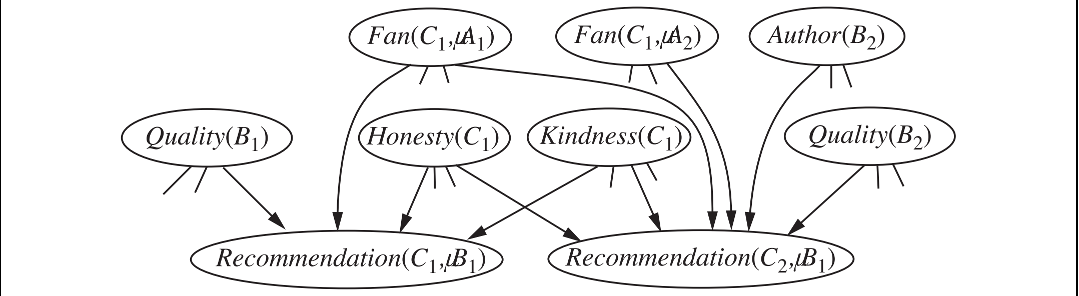


Figure 14.19 Fragment of the equivalent Bayes net when $Author(B_2)$ is unknown.

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:

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

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**. 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.

UNROLLING

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:

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

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 reasoning 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 $A \lor B$ 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:

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

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

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

These two rules form a feedback loop: evidence for Rain increases the belief in $\mathit{WetGrass}$, 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

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

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 $\mathit{Sprinkler} \mapsto \mathit{Rain}$.

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**, written $Bel(X)$.

BELIEF FUNCTION

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.

MASS

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 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 $\mathbf{P}(\mathit{Flip} \mid \mathit{Bias})$ simply defines how the bias operates. If $\mathbf{P}(\mathit{Bias})$ is symmetric about 0.5, then our prior probability for the flip is

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

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 $\mathit{Tall}(\mathit{Nate})$ 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 $\mathit{Tall}(\mathit{Nate}) \wedge \mathit{Heavy}(\mathit{Nate})$ 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

$$\begin{aligned} &T(A \wedge B) = \min(T(A), T(B)) \\ &T(A \vee B) = \max(T(A), T(B)) \\ &T(\neg A) = 1 - T(A)\ . \end{aligned}$$

Fuzzy logic is therefore a truth-functional system—a fact that causes serious difficulties. For example, suppose that $T(\mathit{Tall}(\mathit{Nate})) = 0.6$ and $T(\mathit{Heavy}(\mathit{Nate})) = 0.4$. Then we have $T(\mathit{Tall}(\mathit{Nate}) \wedge \mathit{Heavy}(\mathit{Nate})) = 0.4$, which seems reasonable, but we also get the result $T(\mathit{Tall}(\mathit{Nate}) \wedge \neg \mathit{Tall}(\mathit{Nate})) = 0.4$, 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