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 gave the syntax and semantics of probability theory. We remarked on 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 atomic events is rather unnatural and can be very difficult unless a large amount of data is available from which to gather statistical estimates.

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 and to give a concise specification of any full joint probability distribution.

BAYESIAN NETWORK

[1] This is the most common name, but there are many others, 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** will be 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. A set of random variables makes up the nodes of the network. Variables 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 aparent of Y.
3. Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.
4. The graph has no directed cycles (and hence is a directed, acyclic graph, or DAG).

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 in a properly constructed network is usually that X has a direct *influence* on Y. 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, Cavzty, 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 *Cavsty* 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 always calls

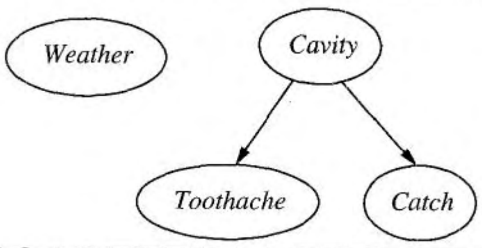


Figure 14.1 A simple Bayesian network in which *Weather* is independent of the other three variables and *Toothache* and *Catch* are conditionally independent, given ***Cavity.***

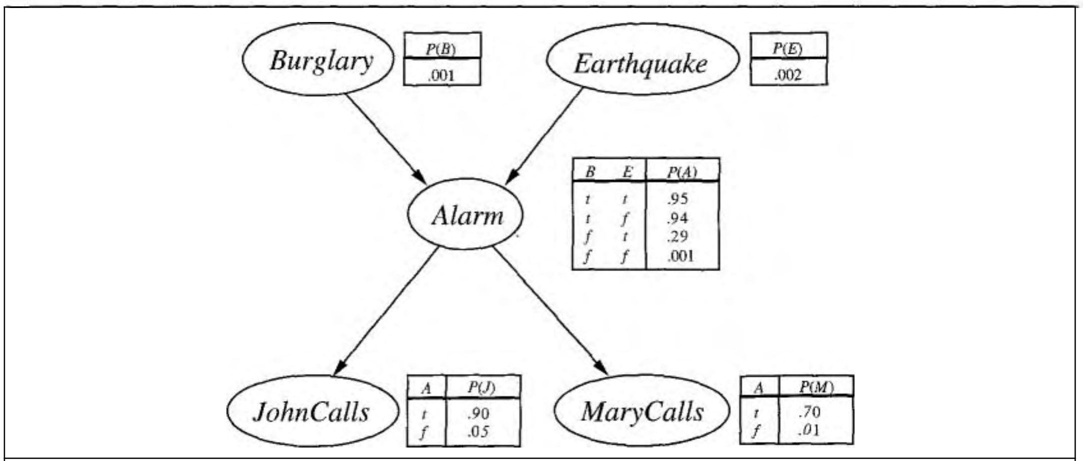


Figure 14.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and sometimes misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary. The Bayesian network for this domain appears in Figure 14.2.

For the moment, let us ignore the conditional distributions in the figure and concentrate on the topology of the network. In the case of the burglary network, the topology 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 any burglaries directly, they do not notice the minor earthquakes, and they do not confer before calling.

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.

CONDITIONAL PROBABILITY TABLE

Now let us turn to the conditional distributions shown in Figure 14.2. In the figure, each distribution is 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

CONDITIONING CASE

combination of values for the parent nodes—a miniature atomic event, 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.

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.

Representing the full joint distribution

A Bayesian network provides a complete description of the domain. Every entry in the full joint probability distribution (hereafter abbreviated as "joint") can be calculated from the information in the network. A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1$ $A \ldots \wedge X$, = x,). We use the notation $P(x_1, \ldots,$ *x,)* as an abbreviation for this. The value of this entry is given by the formula

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

where $parents(X_i)$ denotes the specific values of the variables in $Parents(X_i)$. 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. The CPTs therefore provide a decomposed representation of the joint distribution.

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 use single-letter names for the variables:

$$\begin{aligned} &P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) \\ &\quad = P(j|a)P(m|a)P(a|\neg b\ A\ \neg e)P(\neg b)P(\neg e) \\ &\quad = 0.90\ x\ 0.70\ \text{x}\ 0.001\ \text{x}\ 0.999\ \text{x}\ 0.998 = 0.0006'2. \end{aligned}$$

Section 13.4 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 (14.1) defines what a given Bayesian network means. It does not, however, 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 (14.1) 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 joint distribution in terms of a conditional probability, using the product rule (see Chapter 13):

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

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

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

CHAIN RULE

This identity holds true for any set of random variables and is called the **chain rule.** Comparing it with Equation (14.1), we see that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i in the network,

$$\mathbf{P}(X_i|X_{i-1}, \ldots, X_1) = \mathbf{P}(X_i|Parents(X_i)) , \tag{14.2}$$

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

What Equation (14.2) says is that the Bayesian network is a correct representation of the domain only if each node is conditionally independent of its predecessors in the node ordering, given its parents. Hence, in order to construct a Bayesian network with the correct structure for the domain, we need to choose parents for each node such that this property holds. 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:

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

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 very general property of **locally structured** (also called **sparse)** systems. In a locally structured system, each subcomponent interacts directly with only a bounded number of other components, regardless of the total number of components. Local

LOCALLY STRUCTURED

SPARSE

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 very 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.

Even in a locally structured domain, constructing a locally structured Bayesian network is not a trivial problem. We require not only that each variable be directly influenced by only a few others, but also that the network topology actually reflect those direct influences with the appropriate set of parents. Because of the way that the construction procedure works, the "direct influencers" will have to be added to the network first if they are to become parents of the node they influence. Therefore, *the correct order in which to add nodes is to add the "root causes" first, then the variables they influence,* and so on, until we reach the "leaves," which have no direct causal influence on the other variables.

What happens if we happen to choose the wrong order? Let us consider the burglary example again. Suppose we decide to add the nodes in the order $MaryCalls$, JohnCalls, Alarm, Burglary, Earthquake. Then we 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 call, 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:

$$\mathbf{P}(Burglary|Alarm, \text{JohnCalls}, MaryCalls) = \mathbf{P}(Burglary|Alarm)$$

  Hence we need just Alarm as parent.

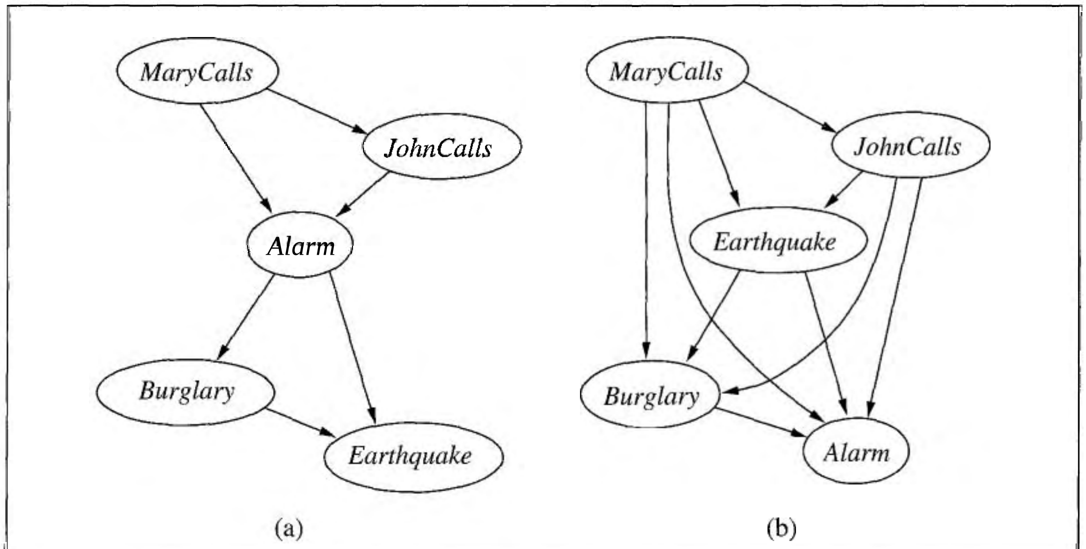


Figure 14.3 Network structure depends on order of introduction. In each network, we have introduced nodes in top-to-bottom order.

- 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 Chapter 8. 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 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.

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 (114.1). 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 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 these we can derive the "numerical" semantics. The topological semantics is given by either of the following specifications, which are equivalent:[2]

DESCENDANTS

1. A node is conditionally independent of its non-descendants, given its parents. For example, in Figure 14.2, *JohnCalls* is independent of *.Burglary* and *Earthquake*, given the value of *Alarm.*

MARKOV BLANKET

2. 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. For example, *Burglary* is independent of *JohnCalls* and *MaryCalls*, given *Alarm* and *Earthquake*.

These specifications are illustrated in Figure 14.4. From these conditional independence assertions and the CPTs, the full joint distribution can be reconstructed; thus, the "numerical" semantics and the "topological" semantics are equivalent.

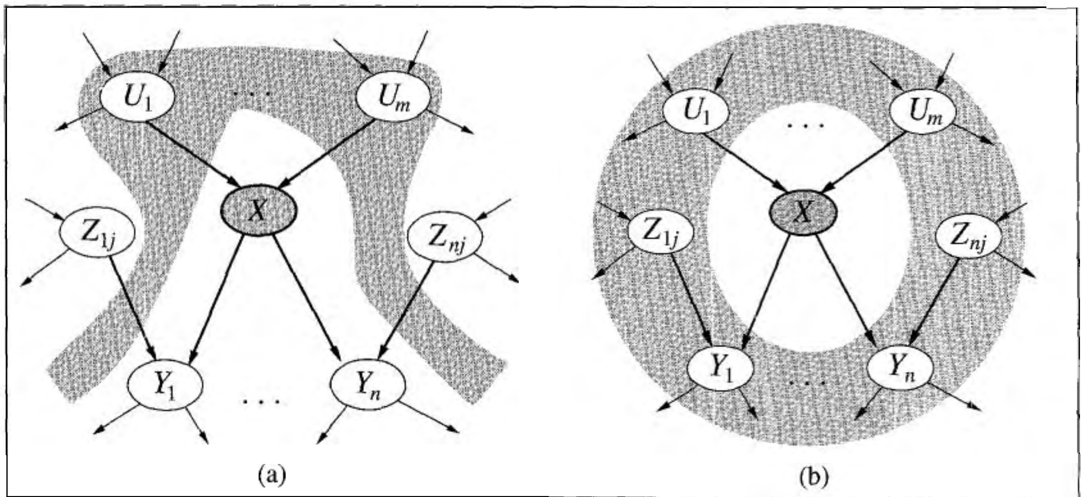


Figure 14.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the gray area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the gray area).

[2] There is also a general topological criterion called **d-separation** for deciding whether a set of nodes X is 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 Russell and Norvig (1995) or Pearl (1988). Shachter (1998) gives a more intuitive method of ascertaining d-separation.

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 the inflows (rivers, runoff, precipitation) into a lake and the outflows (livers, evaporation, seepage) from the lake and the child is the change in the water level of the lake, then the value of the child is the difference between the inflow parents and the outflow parents.

NOISY-OR

LEAK NODE

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. (This is not as strict as it seems, because 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 isfalse if and only if all its true parents are inhibited, and the probability of this is the product of the inhibition probabilities for each parent. Let us suppose these individual inhibition probabilities are as follows:

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

Then, from this information and the noisy-OR assumptions, the entire CPT can be built. The following table shows how:

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 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 other 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 $N(\mu, a^2)(z)$ has the mean μ and the variance σ^2 as parameters.

DISCRETIZATION

PARAMETERS

HYBRID BAYESIAN NETWORK

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 *Bugs* is discrete and has a continuous parent.

For the *Cost* variable, we need to specify $\mathbf{P}(Cost|\,Harvest, Subsidy)$. The discrete parent is handled by explicit enumeration—that is, specifying both $P(Cost|Harvest, .subsidy)$ and $P(Cost|Harvest, \neg subsidy)$. 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

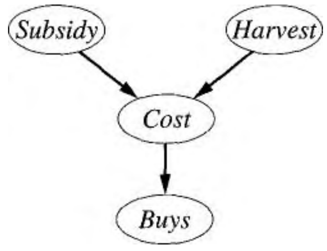


Figure 14.5 A simple network with discrete variables (Subsidy and Buys) and continuous variables (Harvest and Cost).

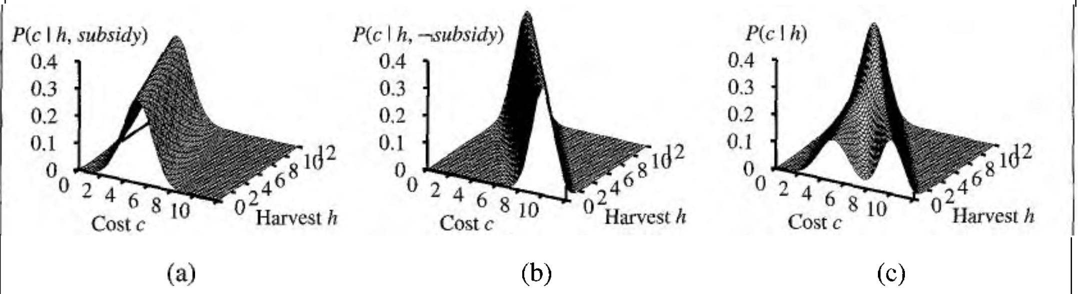


Figure 14.6 The graphs in (a) and (b) show the probability distribution over Cost as a function of Harvest size, with Subsidy true and false respectively. Graph (c) shows the distribution $P(Cost|\text{Harvest})$, obtained by summing over the two subsidy cases.

LINEAR GAUSSIAN **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 a is fixed. We need two distributions, one for *subsidy* and one for $\neg subsidy$, with different parameters:

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

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

For this example, then, the conditional distribution for *Cost* is specified by naming the linear Gaussian distribution and providing the parameters a_t, b_t, σ_t, a_f, b_f, and of. Figures 14.6(a) and (b) show these two relationships. Notice that in each case the slope is negative, because price decreases as supply increases. (Of course, the assumption of linearity implies that the price 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 $P(c|\text{h})$, 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 distribu-

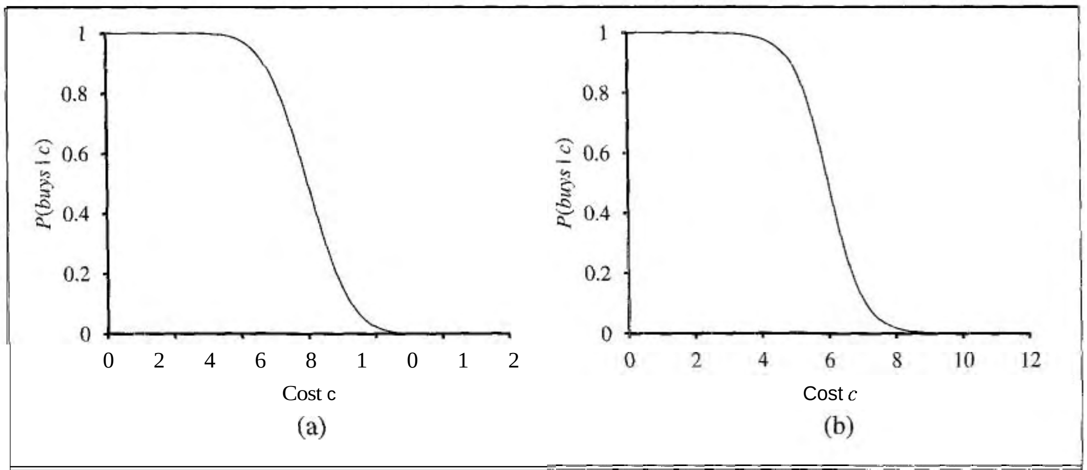


Figure 14.7 (a) A probit distribution for the probability of *Buys* given *Cost*, with $\mu = 6.0$ and $a = 1.0$. (b) A logit distribution with the same parameters.

tion that is a multivariate Gaussian distribution over all the variables (Exercise 14.5).[3] (A multivariate Gaussian distribution is a surface in more than one dimension that has a peak at the mean (in n dimensions) and drops off on all sides.) When discrete variables are added (provided that no discrete variable is a child of a continuous variable), 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 a, and the probability of buying decreases as cost increases.

PROBIT DISTRIBUTION

This **probit distribution** 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. An alternative to the probit model is the **logit distribution,** which uses the **sigmoid function** to produce a soft threshold:

LOGIT DISTRIBUTION

SIGMOID FUNCTION

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

[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 will see that inference for networks of discrete variables is NP-hard.

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. Extensions for a multivalued discrete child are explored in Exercise 14.6.

14.4 Exact Inference in Bayesian Networks

EVENT

HIDDEN VARIABLES

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. We will use the notation introduced in 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 denote the nonevidence variables $Y_1, \ldots, Y_l$ (sometimes called the hidden variables). Thus, the complete set of variables $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$. A typical query asks for the posterior probability distribution $\mathbf{P}(X|\mathbf{e})$.[4]

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:

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

In this section we will 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.

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 $\mathbf{P}(X|\mathbf{e})$ can be answered using Equation (13.6), which we repeat here for convenience:

$$\mathbf{P}(X|\mathbf{e}) = a\,\mathbf{P}(X, \mathbf{e}) = a \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) .$$

Now, a Bayesian network gives a complete representation of the full joint distribution. More specifically, Equation (14.1) 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.*

In Figure 13.4, an algorithm, ENUMERATE-JOINT-ASK, was given for inference by enumeration from the full joint distribution. The algorithm takes as input a full joint distribution $\mathbf{P}$ and looks up values therein. It is a simple matter to modify the algorithm so that it takes

[4] We will assume that the query variable is not among thc evidence variables; if it is, then the posterior distribution for X simply gives probability 1 to the observed value. For simplicity, we have also assumed that the query is just a single variable. Our algorithms can be extended easily to handle a joint query over several variables.

as input a Bayesian network bn and "looks up" joint entries by multiplying the corresponding CPT entries from bn.

Consider the query $\mathbf{P}(Burglary|JohnCalls = \text{true}, MaryCalls = \text{true})$. The hidden variables for this query are Earthquake and Alarm. From Equation (13.6), using initial letters for the variables in order to shorten the expressions, we have[5]

$$\mathbf{P}(B|j,m) = \alpha\,\mathbf{P}(B,j,m) = \alpha \sum_e \sum_a \mathbf{P}(B,e,a,j,m)\,.$$

The semantics of Bayesian networks (Equation (14.1)) then gives us an expression in terms of CPT entries. For simplicity, we will do this just for $Burglary = \text{true}$:

$$P(b|j,m) = \alpha \sum_e \sum_a P(b)P(e)P(a|b,e)P(j|a)P(m|a)$$

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

$$P(b|j,m) = \alpha\,P(b) \sum_e P(e) \sum_a P(a|b,e)P(j|a)P(m|a)\,. \qquad (14.3)$$

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

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

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

The evaluation process for the expression in Equation (14.3) 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. Thus, the space complexity of ENUMERATION-ASK is only linear in the number of variables—effectively, 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 $O(n2^n)$ for the simple approach described earlier, but still rather grim. One thing to note about the tree in Figure 14.8 is that it makes explicit the *repeated subexpressions* that are evaluated by the algorithm. The products $P(j|a)P(m|a)$ and $P(j|\neg a)P(m|\neg a)$ are computed twice, once for each value of e. The next section describes a general method that avoids such wasted computations.

[5] An expression such as $\sum_e P(a,e)$ means to sum $P(A = a, E = e)$ for *all* possible values of e. 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.

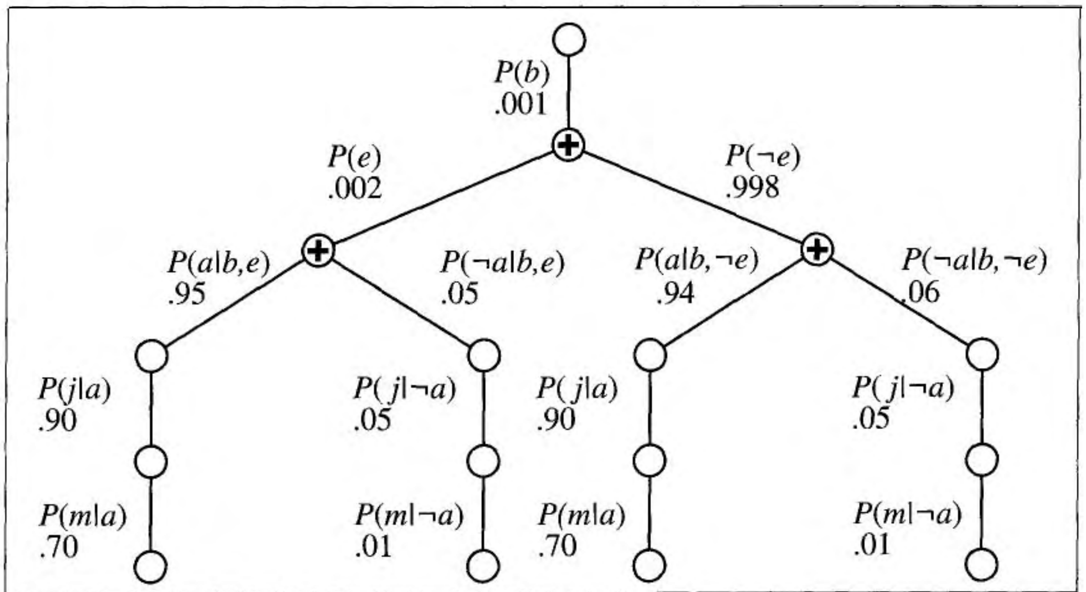


Figure 14.8 The structure of the expression shown in Equation (14.3). The evaluation proceeds top-down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

```
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} ∪ E ∪ Y   /* Y = hidden variables */

  Q($X$) ← a distribution over X, initially empty
  for each value $x_i$ of X do
     extend e with value $x_i$ for X
     Q($x_i$) ← ENUMERATE-ALL(VARS[$bn$], e)
  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 $P(y \mid parents(Y))$ x ENUMERATE-ALL(REST($vars$), e)
     else return $\sum_y P(y \mid parents(Y))$ x 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.

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

Variable elimination works by evaluating expressions such as Equation (14.3) 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.

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

$$\mathbf{P}(B|j,m) = \alpha\,\underbrace{\mathbf{P}(B)}_{B}\sum_{e}\underbrace{P(e)}_{E}\sum_{a}\underbrace{\mathbf{P}(a|B,e)}_{A}\underbrace{P(j|a)}_{J}\underbrace{P(m|a)}_{M}\ .$$

Notice that we have annotated each part of the expression with the name of the associated variable; these parts are called **factors**. The steps are as follows:

FACTORS

- The factor for M, $P(m|a)$, does not require summing over M (because M's value is already fixed). We store the probability, given each value of a, in a two-element vector,

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix}$$

  (The $\mathbf{f}_M$ means that M was used to produce f.)
- Similarly, we store the factor for J as the two-element vector $\mathbf{f}_J(A)$.
- The factor for A is $\mathbf{P}(a|B,e)$, which will be a $2\times2\times2$ matrix $\mathbf{f}_A(A,B,E)$.
- Now we must sum out A from the product of these three factors. This will give us a 2 x 2 matrix whose indices range over just B and E. We put a bar over A in the name of the matrix to indicate that A has been summed out:

$$\begin{aligned}\mathbf{f}_{\bar{A}JM}(B,E) &= \sum_a \mathbf{f}_A(a,B,E)\times\mathbf{f}_J(a)\times\mathbf{f}_M(a)\\ &= \mathbf{f}_A(a,B,E)\times\mathbf{f}_J(a)\times\mathbf{f}_M(a)\\ &+ \mathbf{f}_A(\neg a,B,E)\times\mathbf{f}_J(\neg a)\times\mathbf{f}_M(\neg a)\ .\end{aligned}$$

  POINTWISE PRODUCT

  The multiplication process used here is called a **pointwise product** and will be described shortly.
- We process E in the same way: sum out E from the product of $\mathbf{f}_E(E)$ and $\mathbf{f}_{\bar{A}JM}(B,E)$:

$$\begin{aligned}\mathbf{f}_{\bar{E}\bar{A}JM}(B) &= \mathbf{f}_E(e)\times\mathbf{f}_{\bar{A}JM}(B,e)\\ &+ \mathbf{f}_E(\neg e)\times\mathbf{f}_{\bar{A}JM}(B,\neg e)\ .\end{aligned}$$

- Now we can compute the answer simply by multiplying the factor for B (i.e., $\mathbf{f}_B(B) = \mathbf{P}(B)$), by the accumulated matrix $\mathbf{f}_{\bar{E}\bar{A}JM}(B)$:

$$\mathbf{P}(B|j,m) = \alpha\,\mathbf{f}_B(B)\times\mathbf{f}_{\bar{E}\bar{A}JM}(B)\ .$$

Exercise 14.7(a) asks you to check that this process yields the correct answer.

Examining this sequence of steps, we see that there are two basic computational operations required: pointwise product of a pair of factors, and sumrning out a variable from a product of factors.

The pointwise product is not matrix multiplication, nor is it element-by-element multiplication. The pointwise product of two factors $\mathbf{f}_1$ and $\mathbf{f}_2$ yields a new factor f whose variables are the union of the variables in $\mathbf{f}_1$ and $\mathbf{f}_2$. 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 $\mathbf{f}_2(B,C)$ with probability distributions shown below, the pointwise product $\mathbf{f}_1$ x $\mathbf{f}_2$ is given as $\mathbf{f}_3(A,B,C)$:

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	$.3 \times .2$
T	F	.7	T	F	.8	T	T	F	$.3 \times .8$
F	T	.9	F	T	.6	T	F	T	$.7 \times .6$
F	F	.1	F	F	.4	T	F	F	$.7 \times .4$
						F	T	T	$.9 \times .2$
						F	T	F	$.9 \times .8$
						F	F	T	$.1 \times .6$
						F	F	F	$.1 \times .4$

Summing out a variable from a product of factors is also a straightforward computation. 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 process. For example,

$$\textstyle\sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A,B,e) \times \mathbf{f}_J(A) \times \mathbf{f}_M(A) = \\ \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A,B,e)$$

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

$$\mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \sum_e \mathbf{f}_E(e) \times \mathbf{f}_A(A,B,e) = \mathbf{f}_J(A) \times \mathbf{f}_M(A) \times \mathbf{f}_{\bar{E},A}(A,B)\,.$$

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 routines for pointwise product and summing out, the variable elimination algorithm itself can be written quite simply, as shown in Figure 14.10.

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

$$P(J|b) = \alpha\, P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

If we evaluate this expression from right to left, we notice something interesting: $\sum_m P(m|a)$ 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 $P(JohnCalls|\text{Burglary} = \text{true})$ 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

```
function ELIMINATION-ASK($X$, e, bn) returns a distribution over $X$
   inputs: X, the query variable
           e, evidence specified as an event
           bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

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

Figure 14.10 The variable elimination algorithm for answering queries on Bayesian networks.

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.

The complexity of exact inference

We have argued that variable elimination is more efficient than enumeration because it avoids repeated computations (as well as dropping irrelevant variables). 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 is determined by the order of elimination of variables and by the structure of the network.

The burglary network of Figure 14.2 belongs to the family of networks in which there is at most one undirected path between any two nodes in the network. These are called **singly connected** networks or **polytrees,** and they have a particularly nice property: *The time and space complexity of exact inference in polytrees is linear in the size of the network.* I-Iere, 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. These results hold for any ordering consistent with the topological ordering of the network (Exercise 14.7).

SINGLY CONNECTED

POLYTREES

MULTIPLY CONNECTED

For **multiply connected** networks, such as that of Figure 14.11(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 includes inference in propositional logic as a special case, inference in Bayesian networks is NP-hard.* In fact, it can be shown (Exercise 14.8) 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 5, the difficulty of solving a discrete CSP is related to how "tree-like" its constraint graph is.

Measures such as **hypertree 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.

Clustering algorithms

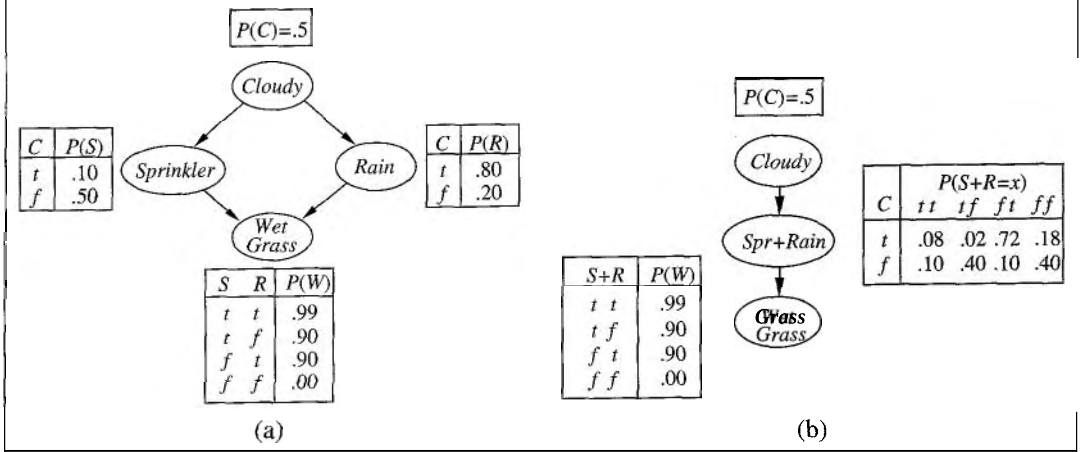
The variable elimination algorithm is simple and efficient for answering individual queries. If we want to compute posterior probabilities for all the variables in a network, however, it can be less efficient. For example, in a polytree network, one would need to issue $O(n)$ queries costing $O(n)$ each, for a total of $0(n^2)$ 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 14.11(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 14.11(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.

Once the network is in polytree form, a special-purpose inference algorithm is applied. Essentially, the algorithm is a form of constraint propagation (see Chapter 5) where the constraints ensure that neighboring clusters 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 $O(n)$, where n is now the size of the modified 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 require exponential time and space to construct.



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	t t	t f	f t	f f
t	.08	.02	.72	.18
f	.10	.40	.10	.40

Figure 14.11 (a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

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. In recent years, Monte Carlo algorithms have become widely used in computer science to estimate quantities that are difficult to calculate exactly. For example, the simulated annealing algorithm described in Chapter 4 is a Monte Carlo method for optimization problems. 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.

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 *(heads,tails)* 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 in the range $[0,1]$, it is a simple matter to sample any distribution on a single variable. (See Exercise 14.9.)

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 14.12. We can illustrate its operation on the network in Figure 14.11(a), assuming an ordering $[Cloudy, Sprinkler, Rain, WetGrass]$:

1. Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5\rangle$; suppose this returns *true.*
2. Sample from $\mathbf{P}(Sprinkler|Cloudy = true) = \langle 0.1, 0.9\rangle$; suppose this returns *false.*
3. Sample from $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2\rangle$; suppose this returns *true.*
4. Sample from $\mathbf{P}(WetGrass|Sprinkler = false, Rain = true) = \langle 0.9, 0.1\rangle$; suppose this returns *true.*

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

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

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

  $\mathbf{x} \leftarrow$ an event with $n$ elements
  for $i$ = 1 to $n$ do
      $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$
```

Figure 14.12 A sampling algorithm that generates events from a Bayesian network.

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 (14.1). That is, we have

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

This simple fact makes it very 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 $N(x_1, \ldots, x_n)$ be the frequency of the specific event $x_1, \ldots, x_n$. We expect this frequency to converge, in the limit, to its expected value according to the sampling probability:

$$\lim_{N\to\infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) . \qquad (14.4)$$

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

$$P(x_1, \ldots, x_m) \approx N_{PS}(x_1, \ldots, x_m)/N . \qquad (14.5)$$

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 $P(X|\mathbf{e})$. The REJECTION-SAMPLING algorithm is shown in Figure 14.13. First, it generates samples from the prior distribution specified

```
function REJECTION-SAMPLING($X$, e, bn, $N$) returns an estimate of $P(X|\mathbf{e})$
  inputs: $X$, the query variable
          e, evidence specified as an event
          bn, a Bayesian network
          $N$, the total number of samples to be generated
  local variables: N, a vector of counts over $X$, 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[$X$])
```

Figure 14.13 The rejection sampling algorithm for answering queries given evidence in a Bayesian network.

by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate $\hat{P}(X=x|\mathbf{e})$ is obtained by counting how often $X=x$ occurs in the remaining samples.

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

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

From Equation (14.5), this becomes

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

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

Continuing with our example from Figure 14.11(a), let us assume that we wish to estimate $\mathbf{P}(Rain|Sprinkler=true)$, 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,

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

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 $\mathbf{P}(Rain|RedSkyAtNight=true)$, 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

Likelihood weighting avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence *e*. We begin by describing how the algorithm works; then we show that it works correctly—that is, generates consistent probability estimates.

LIKELIHOOD-WEIGHTING (see Figure 14.14) fixes the values for the evidence variables E and samples only the remaining variables X and Y. 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 14.11(a), with the query $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$. The process goes as follows: First, the weight w is set to 1.0. Then an event is generated:

1. Sample from $\mathbf{P}(Cloudy) = \langle 0.5, 0.5\rangle$; suppose this returns *true*.
2. *Sprinkler* is an evidence variable with value *true*. Therefore, we set
$$w \leftarrow w \times P(Sprinkler = true|Cloudy = true) = 0.1 .$$
3. Sample from $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2\rangle$; suppose this returns *true*.
4. *WetGrass* is an evidence variable with value *true*. Therefore, we set
$$w \leftarrow w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099 .$$

Here WEIGHTED-SAMPLE returns the event *[true,true, true, true]* with weight 0.099, and this is tallied under $Rain = true$. The weight is low because the event describes a cloudy day, which makes the sprinkler unlikely to be on.

To understand why likelihood weighting works, we start by examining the sampling distribution S_{WS} for WEIGHTED-SAMPLE. Remember that the evidence variables E are fixed with values *e*. We will call the other variables Z, that is, $Z = \{X\} \cup Y$. The algorithm samples each variable in Z given its parent values:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i|parents(Z_i)) . \qquad (14.6)$$

Notice that $Parents(Z_i)$ can include both hidden 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. On the other hand, S_{WS} pays less attention to the evidence than does the true posterior distribution $P(\mathbf{z}|\mathbf{e})$, because the sampled values for each Z_i *ignore* evidence among Z_i's non-ancestors.[6]

[6] Ideally, we would like to use a sampling distribution equal to the true posterior $P(\mathbf{z}|\mathbf{e})$, 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.

```
function LIKELIHOOD-WEIGHTING($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $P(X|\mathbf{e})$
  inputs: $X$, the query variable
          $\mathbf{e}$, evidence specified as an event
          $bn$, a Bayesian network
          $N$, the total number of samples to be generated
  local variables: $\mathbf{W}$, a vector of weighted counts over $X$, initially zero

  for $j$ = 1 to $N$ do
      $\mathbf{x}, w \leftarrow$ WEIGHTED-SAMPLE($bn$, $\mathbf{e}$)
      $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where $x$ is the value of $X$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{W}[X]$)
```

```
function WEIGHTED-SAMPLE($bn$, $\mathbf{e}$) returns an event and a weight

  $\mathbf{x} \leftarrow$ an event with $n$ elements; $w \leftarrow 1$
  for $i$ = 1 to $n$ do
      if $X_i$ has a value $x_i$ in $\mathbf{e}$
          then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
          else $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$, $w$
```

Figure 14.14 The likelihood weighting algorithm for inference in Bayesian networks.

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):

$$w(\mathbf{z},\mathbf{e}) = \prod_{i=1}^{m} P(e_i|parents(E_i)) . \qquad (14.7)$$

Multiplying Equations (14.6) and (14.7), we see that the weighted probability of a sample has the particularly convenient form

$$\begin{aligned} S_{WS}(\mathbf{z},\mathbf{e})w(\mathbf{z},\mathbf{e}) &= \prod_{i=1}^{l} P(z_i|parents(Z_i)) \prod_{i=1}^{m} P(e_i|parents(E_i)) \\ &= P(\mathbf{z},\mathbf{e}) , \end{aligned} \qquad (14.8)$$

because the two products cover all the variables in the network, allowing us to use Equation (14.1) 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:

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

$$= \alpha' \sum_{\mathbf{y}} P(x, \mathbf{y}, \mathbf{e}) \qquad \text{by Equation (14.8)}$$
$$= \alpha' P(x, \mathbf{e}) = P(x|\mathbf{e}) .$$

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 samples will be simulations that bear little resemblance to the reality suggested by the evidence.

Inference by Markov chain simulation

MARKOV CHAIN MONTE CARLO

In this section, we describe the **Markov chain Monte Carlo** (MCMC) algorithm for inference in Bayesian networks. We will first describe what the algorithm does, then we will explain why it works and why it has such a complicated name.

The MCMC algorithm

Unlike the other two sampling algorithms, which generate each event from scratch, MCMC generates each event by making a random change to the preceding event. It is therefore helpful to think of the network as being in a particular *current state* specifying a value for every variable. The next state is generated by randomly sampling a value for one of the nonevidence variables X_i, *conditioned on the current values of the variables in the Markov blanket of* X_i. (Recall from page 499 that the Markov blanket of a variable consists of its parents, children, and children's parents.) MCMC 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 $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$ applied to the network in Figure 14.11(a). The evidence variables *Sprinkler* and *WetGrass* are fixed to their observed values and the hidden 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 following steps are executed repeatedly:

1. *Cloudy* is sampled, given the current values of its Markov blanket variables: in this case, we sample from $\mathbf{P}(Cloudy|Sprinkler = true, Rain = false)$. (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 $\mathbf{P}(Rain|Cloudy = false, Sprinkler = true, WetGrass = true)$. 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

```
function MCMC-ASK($X$, e, bn, N) returns an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{N}[X]$, a vector of counts over X, initially zero
                   Z, the nonevidence variables in bn
                   x, the current state of the network, initially copied from e

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

Figure 14.15 The MCMC algorithm for approximate inference in Bayesian networks.

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 14.15.

Why MCMC works

We will now show that MCMC 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

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 will 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 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 $\mathbf{x}'$:

MARKOV CHAIN

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

We will 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

STATIONARY DISTRIBUTION

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') \qquad \text{for all } \mathbf{x}' . \qquad (14.9)$$

Under certain standard assumptions about the transition probabilrty distribution q,[7] there is exactly one distribution π satisfying this equation for any given q.

[7] The Markov chain defined by q must be ergodic—thatis, essentially, every state must be reachable from every other, and there can be no strictly periodic cycles.

Equation (14.9) 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. This is the property of ***detailed balance:***

DETAILED BALANCE

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

We can show that detailed balance implies stationarity simply by summing over $\mathbf{x}$ in Equation (14.10). We have

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

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

Now we will show that the transition probability $q(\mathbf{x} \to \mathbf{x}')$ defined by the sampling step in MCMC-ASK satisfies the detailed balance equation with a stationary distribution equal to $P(\mathbf{x}|\mathbf{e})$, (the true posterior distribution on the hidden variables). We will do this in two steps. First, we will define a Markov chain in which each variable is sampled conditionally on the current values of *all* the other variables, and we will show that this satisfies detailed balance. Then, we will simply observe that, for Bayesian networks, doing that is equivalent to sampling conditionally on the variable's Markov blanket (see page 499).

Let X_i be the variable to be sampled, and let $\overline{\mathbf{X}_i}$ be all the hidden variables ***other** than* X_i. Their values in the current state are x_i and $\overline{\mathbf{x}_i}$. If we sample a new value x_i' for X_i conditionally on all the other variables, including the evidence, we have

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

GIBBS SAMPLER

This transition probability is called the ***Gibbs sampler*** and is a particularly convenient form of MCMC. Now we show that the Gibbs sampler is in detailed balance with the true posterior:

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

As stated on page 499, a variable is independent of all other variables given its Markov blanket; hence,

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

where $mb(X_i)$ denotes the values of the variables in X_i's Markov blanket, $MB(X_i)$. As shown in Exercise 14.10, the probability of a variable given its Markov blanket is proportional to the probability of the variable given its parents times the probability of each child given its respective parents:

$$P(x_i'|mb(X_i)) = \alpha\, P(x_i'|parents(X_i)) \times \prod_{Y_j \in Children(X_i)} P(y_j|parents(Y_j)) . \tag{14.11}$$

Hence, to flip each variable X_i, the number of multiplications required is equal to the number of X_i's children.

We have discussed here only one simple variant of MCMC, namely the Gibbs sampler. In its most general form, MCMC is a powerful method for computing with probability models and many variants have been developed, including the simulated annealing algorithm presented in Chapter 4, the stochastic satisfiability algorithms in Chapter 7, and the Metropolis–Hastings sampler in Chapter 15.

14.6 Extending Probability to First-Order Representations

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 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 offirst-order representations, we expect to be able to increase dramatically the range ofproblems that can be handled.*

The basic insight required to achieve this goal is the following: In the propositional context, a Bayesian network specifies probabilities over atomic events, each of which specifies a value for each variable in the network. Thus, an atomic event is a **model** or **possible world,** in the terminology of propositional logic. In the first-order context, a model (with its interpretation) specifies a domain of objects, the relations that hold among those objects, and a mapping from the constants and predicates of the knowledge base to the objects and relations in the model. Therefore, *a first-order probabilistic knowledge base should specify probabilities for all possible first-order models.* Let $\mu(M)$ be the probability assigned to model M by the knowledge base. For any first-order sentence ϕ, the probability $P(\phi)$ is given in the usual way by summing over the possible worlds where ϕ is true:

$$P(\phi) = \sum_{M:\phi \text{ is true in } M} \mu(M) . \tag{14.12}$$

So far, so good. There is, however, a problem: the set of first-order models is infinite. This means that (I) the summation could be infeasible, and (2) specifying a complete, consistent distribution over an infinite set of worlds could be very difficult.

Let us scale back our ambition, at least temporarily. In particular, let us devise a restricted language for which there are only finitely many models of interest. There are several ways to do this. Here, we present **relational probability models,** or RPMs, which borrow ideas from semantic networks (Chapter 10) and from object-relational databases. Other approaches are discussed in the bibliographical and historical notes.

RELATIONAL PROBABILITY MODEL

RPMs allow constant symbols that name objects. For example, let *ProfSmith* be the name of a professor, and let *Jones* be the name of a student. Each object is an instance of a class; for example, *ProfSmith* is a *Professor* and *Jones* is a *Student*. We assume that the class of every constant symbol is known.

SIMPLE FUNCTION

Our function symbols will be divided into two kinds. The first kind, simple functions, maps an object not to another structured object, but to a value from a fixed domain of values, just like a random variable. For example, $Intelligence(Jones)$ and $Funding(ProfSmith)$ might be *hi* or *lo*; $Success(Jones)$ and $Fame(ProfSmith)$ may be *true* or *false.* Function symbols must not be applied to values such as *true* and *false*, so it is not possible to have nesting of simple functions. In this way, we avoid one source of infinities. The value of a simple function applied to a given object may be observed or unknown; these will be the basic random variables of our representation.[8]

COMPLEX FUNCTION

We also allow complex functions, which map objects to other objects. For example, $Advisor(Jones)$ may be $ProfSmith$. Each complex function has a specified domain and range, which are classes. For example, the domain of *Advisor* is *Student* and the range is *Professor.* Functions apply only to objects of the right class; for instance, the *Advisor* of $ProfSmith$ is undefined. Complex functions may be nested: $DeptHead(Advisor(Jones))$ could be $ProfMoore$. We will assume (for now) that the values of all complex functions are known for all constant symbols. Because the KB is finite, this implies that every chain of complex function applications leads to one of a finite number of objects.[9]

The last element we need is the probabilistic information. For each simple function, we specify a set of parents, just as in Bayesian networks. The parents can be other simple functions of the same object; for example, the *Funding* of a *Professor* might depend on his or her *Fame.* The parents can also be simple functions of related objects—for example, the *Success* of a student could depend on the *Intelligence* of the student and the *Fame* of the student's advisor. These are really *universally* quantified assertions about the parents of all the objects in a class. Thus, we could write

$$\forall x \ \ x \in Student \Rightarrow$$
$$Parents(Success(x)) = \{Intelligence(x), Fame(Advisor(x))\} \ .$$

(Less formally, we can draw diagrams like Figure 14.16(a).) Now we specify the conditional probability distribution for the child, given its parents. For example, we might say that

$$\forall x \ \ x \in Student \Rightarrow$$
$$P(Success(x) = true | Intelligence(x) = hi, Fame(Advisor(x)) = true) = 0.95$$

Just as in semantic networks, we can attach the conditional distribution to the class itself, so that the instances inherit the dependencies and conditional probabilities from the class.

The semantics for the RPM language assumes that every constant symbol refers to a distinct object—the unique names assumption described in Chapter 10. Given this assumption and the restrictions listed previously, it can be shown that every RPM generates a fixed, finite set of random variables, each of which is a simple function applied to a constant symbol. Then, provided that the parent–child dependencies are acyclic, we can construct an equivalent Bayesian network. That is, the RPM and the Bayesian network specify identical probabili-

[8] They play a role very similar to that of the ground atomic sentences generated in the propositionalization process described in Section 9.1.

[9] This restriction means that we cannot use complex functions such as Father and *Mother,* which lead to potentially infinite chains that would have to end with an unknown object. We revisit this restriction later.

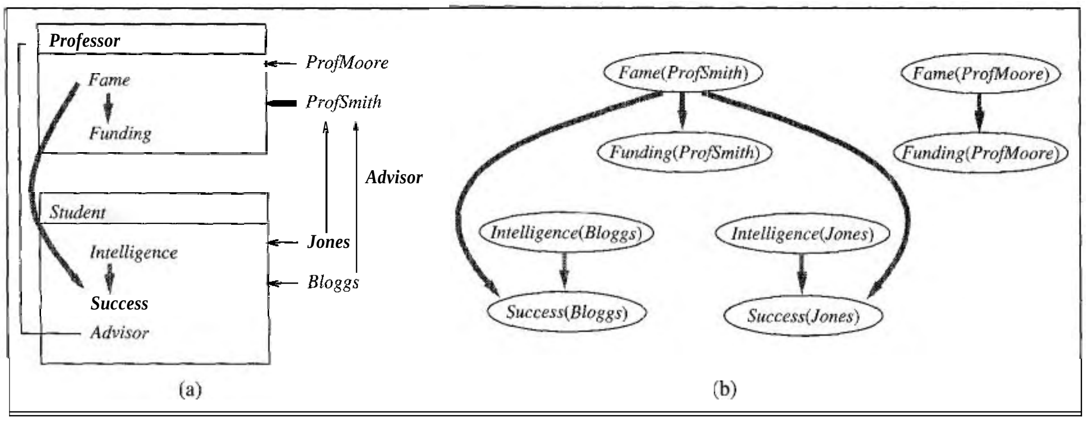


Figure 14.16 (a) An RPM describing two classes: *Professor* and *Student*. There are two professors and two students, and *ProfSmith* is the advisor of both students. (b) The Bayesian network equivalent to the RPM in (a).

ties for each possible world. Figure 14.16(b) shows the Bayesian network corresponding to the RPM in Figure 14.16(a). Notice that the *Advisor* links in the RPM are absent in the Bayesian network. This is because they are fixed and known. They appear implicitly in the network topology, however; for example, *Success*(*Jones*) has *Fame*(*ProfSmith*) as a parent because *Advisor*(*Jones*) is *ProfSmith*. In general, the relations that hold among the objects determine the pattern of dependencies among the properties of those objects.

There are several ways to increase the expressive power of RPMs. We can allow **recursive dependencies** among variables to capture certain kinds of recurring relationships. For example, suppose that addiction to fast food is caused by the *McGene*. Then, for any x, *McGene*(x) depends on *McGene*(*Father*(x)) and *McGene*(*Mother*(x)), which depend in turn on *McGene*(*Father*(*Father*(x))), *McGene*(*Mother*(*Father*(x))), and so on. Even though such knowledge bases correspond to Bayesian networlks with infinitely many random variables, solutions can sometimes be obtained from fixed-point equations. For example, the equilibrium distribution of the *McGene* can be calculated, given the conditional probability of inheritance. Another very important family of recursive knowledge bases consists of the temporal probability models described in Chapter 15. In these models, properties of the state at time t depend on properties of the state at time $t - 1$, and so on.

RECURSIVE DEPENDENCY

RELATIONAL UNCERTAINTY

RPMs can also be extended to allow for relational uncertainty—that is, uncertainty about the values of complex functions. For example, we may not know who *Advisor*(*Jones*) is. *Advisor*(*Jones*) then becomes a random variable, with possible values *ProfSmith* and *ProfMoore*. 'The corresponding network is shown in Figure 14.17.

IDENTITY UNCERTAINTY

There can also be identity uncertainty; for example, we might not know whether *Mary* and *ProfSmith* are the same person. With identity uncertainty, the number of objects and propositions can vary across possible worlds. A world where *Mary* and *ProfSmith* are the same person has one fewer object than a world in which they are different people. This makes the inference process more complicated, but the basic principle established in Equation (14.12) still holds: the probability of any sentence is well defined and can be calculated.

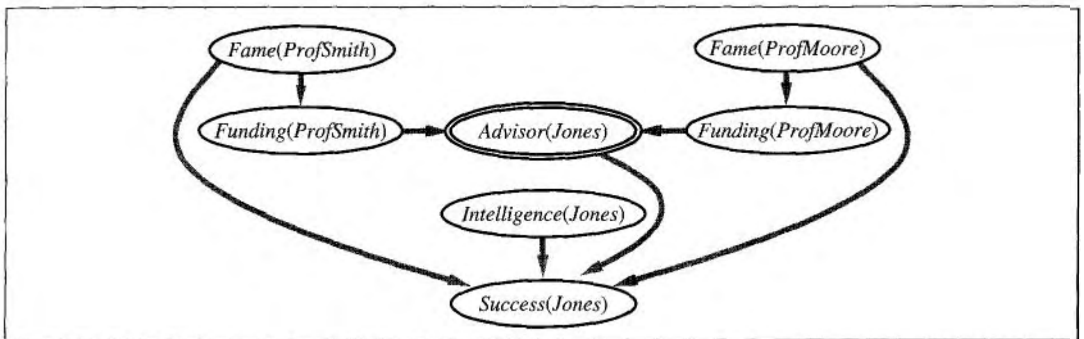


Figure 14.17 Part of the Bayesian network corresponding to an RPM in which *Advisor(Jones)* is unknown, but is either *ProfSmith* or *ProfMoore*. The choice of advisor depends on how much funding each professor has. Notice that *Success(Jones)* will now depend on the *Fame* of *both* professors, although the value of *Advisor(Jones)* determines which one actually has an influence.

Identity uncertainty is particularly important for robots and for embedded sensor systems that must keep track of multiple objects. We return to this problem in Chapter 15.

Let us now examine the question of inference. Clearly, inference can be done in the equivalent Bayesian network, provided that we restrict the RPM language so that the equivalent network is finite and has a fixed structure. This is analogous to the way in which first-order logical inference can be done via propositional inference on the equivalent propositional knowledge base. (See Section 9.1.) As in the logical case, the equivalent network could be too large to construct, let alone evaluate. Dense interconnections are also a problem. (See Exercise 14.12.) Approximation algorithms. such as MCMC (Section 14.5), are therefore very useful for RPM inference.

When MCMC is applied to the equivalent Bayesian network for a simple RPM knowledge base with no relational or identity uncertainty, the algorithm samples from the space of possible worlds defined by the values of simple functions of the objects. It is easy to see that this approach can be extended to handle relational and identity uncertainty as well. In that case, a transition between possible worlds might change the value of a simple function or it might change a complex function, and so lead to a change in the dependency structure. Transitions might also change the identity relations among the constant symbols. Thus, MCMC seems to be an elegant way to handle inference for quite expressive first-order probabilistic knowledge bases.

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 computer vision, natural language understanding, information retrieval, and situation assessment. In all of these areas, the set of objects—and hence the set of random variables—is not known in advance, so purely "propositional" methods, such as Bayesian networks, are incapable of representing the situation completely. They have been augmented by search over the space of model, but RPMs allow reasoning about this uncertainty in a single model.

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 that 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 10.
- Rule-based approaches to uncertainty also have 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 is the only reasonable probability. Obviously, the two cases are different, yet probability seems not to distinguish them. The Deinpster–Shafer theory uses interval-valued degrees of belief to represent an agent's knowledge of the probability of a proposition. Other methods using second-order probabilities are also discussed.

- Probability makes the same ontological commitment as logic: that events 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 an event can be "sort of" true. Vagueness and uncertainty are in fact orthogonal issues, as we will see.

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.

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 in the Markov blanket.

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:

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

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 14.11(a). If we wanted to be able to do both causal and diagnostic reasoning, we would need the two rules

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

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

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

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

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

Given these difficulties, how is it possible that truth-functional systems were ever considered useful? 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. under these circumstances, a minor variation on ce tainty-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. Needless to say, the approach is no longer recommended.

CERTAINTY FACTORS

Representing ignorance: Dempster–Shafer theory

DEMPSTER–SHAFER

BELIEF FUNCTION

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 $Bel(X)$.

We return to coin flipping for an example of belief functions. Suppose a shady character comes up to you and offers to bet you $10 that his coin will come up heads on the next flip. 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. "Dempster's rule" (Dempster, 1968) shows how to combine evidence to give new values for Bel, and Shafer's work extends this into a complete computational model.

As with default reasoning, there is a problem in connecting beliefs to actions. With probabilities, decision theory says that if $P(Heads) = P(\neg Heads) = 0.5$, then (assuming that winning $10 and losing $10 are considered equal magnitude opposites) the reasoner will be indifferent between the action of accepting and declining the bet. A Dempster–Shafer reasoner has $Bel(\neg Heads) = 0$ and thus no reason to accept the bet, but then it also has $Bel(Heads) = 0$ and thus no reason to decline it. Thus, it seems that the Dempster–Shafer reasoner comes to the same conclusion about how to act in this case. Unfortunately, Dempster–Shafer theory allows no definite decision in many other cases where probabilistic inference does yield a specific choice. In fact, the notion of utility in the Dempster–Shafer model is not yet well understood.

One interpretation of Dempster—Shafer theory is that it defines a probability interval: the interval for *Heads* is $[0, 1]$ before our expert testimony and $[0.45, 0.55]$ after. The width of the interval might be an aid in deciding when we need to acquire more evidence: it can tell you that the expert's testimony will help you if you do not know whether the coin is fair, but will not help you if you have already learned that the coin is fair. However, there are no clear guidelines for how to do this, because there is no clear meaning for what the width of an interval means. In the Bayesian approach, this kind of reasoning can be done easily by examining how much one's belief would change if one were to acquire more evidence. For example, knowing whether the coin is fair would have a significant impact on the belief that it will come up heads, and detecting an asymmetric weight would have an impact on the belief that the coin is fair. A complete Bayesian model would include probability estimates for factors such as these, allowing us to express our "ignorance" in terms of how our beliefs would change in the face of future information gathering.

Representing vagueness: Fuzzy sets and fuzzy logic

FUZZY SET THEORY

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

FUZZY LOGIC

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

$$T(A \land B) = \min(T(A), T(B))$$
$$T(A \lor B) = \max(T(A), T(B))$$
$$T(\neg A) = 1 - T(A) .$$

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

FUZZY CONTROL

Fuzzy control is a methodology for constructing control systems in which the mapping between real-valued input and output parameters is represented by fuzzy rules. Fuzzy control has been very successful in commercial products such as automatic transmissions, video cameras, and electric shavers. Critics (see, e.g., Elkan, 1993) argue that these applications are successful because they have small rule bases, no chaining of inferences, and tunable parameters that can be adjusted to improve the system's performance. The fact that they are implemented with fuzzy operators might be incidental to their success; the key is simply to provide a concise and intuitive way to specify a smoothly interpolated, real-valued function.

There have been attempts to provide an explanation of fuzzy logic in terms of probability theory. One idea is to view assertions such as "Nate is Tall" as discrete observations made concerning a continuous hidden variable, Nate's actual Height. The probability model specifies P(Observer says Nate is tall | Height), perhaps using a **probit distribution** as described on page 503. 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(Observer says Nate is tall and heavy | 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 SETS

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

BIBLIOGRAPHICAL AND HISTORICAL NOTES

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

The **influence diagram** representation for decision problems, which incorporated a DAG representation for random variables, was used in decision analysis in the late 1970s (see Chapter 16), but only enumeration was used for evaluation. Judea Pearl developed the message-passing method for carrying out inference in tree networks (Pearl, 1982a) and polytree networks (Kim and Pearl, 1983) and explained the importance of constructing causal rather than diagnostic probability models, in contrast to the certainty-factor systems then in vogue. The first expert system using Bayesian networks was CONVINCE (Kim, 1983; Kim and Pearl, 1987). More recent systems include the MUNIN system for diagnosing neuromuscular disorders (Andersen *et al.*, 1989) and the PATHFINDER system for pathology (Heckerman, 1991). By far the most widely used Bayesian network systems have been the diagnosis-and-repair modules (e.g., the Printer Wizarcl) in Microsoft Windows (Breese and Heckerman, 1996) and the Office Assistant in Microsoft Office (Horvitz *et al.*, 1998).

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

The variable elimination method described in the chapter is closest in spirit to Shachter's method, from which emerged the symbolic probabilistic inference (SPI) algorithm (Shachter *et* al., 1990). SPI attempts to optimize the evaluation of expression trees such as that shown in Figure 14.8. The algorithm we describe is closest to that developed by Zhang and Poole (1994, 1996). Criteria for pruning irrelevant variables were developed by Geiger *et al.* (1990) and by Lauritzen et al. (1990); the criterion we give is a simple special case of these. Rina Dechter (1999) shows how the variable elimination idea is essentially identical to **monserial dynamic programming** (Bertele and Brioschi, 1972), an algorithmic approach that can be applied to solve a range of inference problems in Bayesian networks—for example, finding the **most probable explanation** for a set of observations. This connects Bayesian network algorithms to related methods for solving CSPs and gives a direct measure of the complexity of exact inference in terms of the **hypertree width** of the network.

NONSERIAL DYNAMIC PROGRAMMING

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

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

cHUGIN system (Olesen, 1993). The probit distribution was studied first by Finney (1947), who called it the sigmoid distribution. It has been used widely for modeling discrete choice phenomena and can be extended to handle more than two choices (Daganzo, 1979). Bishop (1995) gives a justification for the use of the logit distribution.

Cooper (1990) showed that the general problem of inference in unconstrained Bayesian networks is NP-hard, and Paul Dagum and Mike Luby (1993) showed the corresponding approximation problem to be NP-hard. Space complexity is also a serious problem in both clustering and variable elimination methods. The method of **cutset conditioning,** which was developed for CSPs in Chapter 5, avoids the construction of exponentially large tables. In a Bayesian network, a cutset is a set of nodes that, when instantiated, reduces the remaining nodes to a polytree that can be solved in linear time and space. The query is answered by summing over all the instantiations of the cutset, so the overall space requirement is still linear (Pearl, 1988). Darwiche (2001) describes a recursive conditioning algorithm that allows a complete range of space/time tradeoffs.

The development of fast approximation algorithms for Bayesian network inference is a very active area, with contributions from statistics, computer science, and physics. The rejection sampling method is a general technique that is long known to statisticians; it was first applied to Bayesian networks by Max Henrion (1988), who called it **logic sampling.** Likelihood weighting, which was developed by Fung and Chang (1989) and Shachter and Peot (1989), is an example of the well-known statistical method of **importance sampling.** A large-scale application of likelihood weighting to medical diagnosis appears in Shwe and Cooper (1991). Cheng and Druzdzel (2000) describe an adaptive version of likelihood weighting that works well even when the evidence has very low prior likelihood.

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

VARIATIONAL APPROXIMATION

There are two very important families of approximation methods that we did not cover in the chapter. The first is the family of **variational approximation** methods, which can be used to simplify complex calculations of all kinds. The basic idea is to propose a reduced version of the original problem that is simple to work with, but that resembles the original problem as closely as possible. The reduced problem is described by some **variational parameters λ** that are adjusted to minimize a distance function D between the original and the reduced problem, often by solving the system of equations $\partial D/\partial\lambda = 0$. In many cases, strict upper and lower bounds can be obtained. Variational methods have long been used in statistics (Rustagi, 1976). In statistical physics, the **mean field** method is a particular variational approximation in which the individual variables making up the model are assumed to be completely independent. This idea was applied to solve large undirected Markov networks (Peterson and Anderson, 1987; Parisi, 1988). Saul *et al.* (1996) developed the mathematical foundations for applying variational methods to Bayesian networks and obtained

VARIATIONAL PARAMETERS

MEAN FIELD

accurate lower-bound approximations for sigmoid networks with the use of mean-field methods. Jaakkola and Jordan (1996) extended the methodology to obtain both lower and upper bounds. Variational approaches are surveyed by Jordan *et al.* (1999).

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

BELIEF PROPAGATION

TURBO DECODING

The connection between probability and first-order languages was first studied by Carnap (1950). Gaifman (1964) and Scott and Krauss (1966) defined a language in which probabilities could be associated with first-order sentences and for which models were probability measures on possible worlds. Within AI, this idea was developed for propositional logic by Nilsson (1986) and for first-order logic by Halpern (199'0). The first extensive investigation of knowledge representation issues in such languages was carried out by Bacchus (1990), and the paper by Wellman *et al.* (1992) surveys early implementation approaches based on the construction of equivalent propositional Bayesian networks. More recently, researchers have come to understand the importance of *complete* knowledge bases—that is, knowledge bases that, like Bayesian networks, define a unique joint distribution over all possible worlds. Methods for doing this have been based on probabilistic versions of logic programming (Poole, 1993; Sato and Kameya, 1997) or semantic networks (Koller and Pfeffer, 1998). Relational probability models of the kind described in this chapter are investigated in depth by Pfeffer (2000). Pasula and Russell (2001) examine both issues of relational and identity uncertainty within RPMs and the use of MCMC inference.

As explained in Chapter 13, early probabilistic systems fell out of favor in the early 1970s, leaving a partial vacuum to be filled by alternative methods. Certainty factors were invented for use in the medical expert system MYCIN (Shortliffe, 1976), which was intended both as an engineering solution and as a mode1 of human judgment under uncertainty. The collection *Rule-Based Expert Systems* (Buchanan and Shortliffe, 1984) provides a complete overview of MYCIN and its descendants (see also Stefik, 1995). David Heckerman (1986) showed that a slightly modified version of certainty factor calculations gives correct probabilistic results in some cases, but results in serious overcounting of evidence in other cases. The PROSPECTOR expert system (Duda *et al.,* 1979) used a rule-based approach in which the rules were justified by a (seldom tenable) global independence assumption.

Dempster–Shafer theory originates with a paper by Arthur Dempster (1968) proposing a generalization of probability to interval values and a combination rule for using them. Later work by Glenn Shafer (1976) led to the Dempster-Shafer theory's being viewed as a competing approach to probability. Ruspini *et al.* (1992) analyze the relationship between the Dempster-Shafer theory and standard probability theory. Shenoy (1989) has proposed a method for decision making with Dempster–Shafer belief functions.

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

The resurgence of probability depended mainly on the discovery of Bayesian networks as a method for representing and using conditional independence information. This resurgence did not come without a fight; Peter Cheeseman's (1985) pugnacious "In Defense of Probability," and his later article "An Inquiry into Computer Understanding" (Cheeseman, 1988, with commentaries) give something of the flavor of the debate. One of the principal objections of the logicists was that the numerical calculations that probability theory was thought to require were not apparent to introspection and presumed an unrealistic level of precision in our uncertain knowledge. The development of qualitative probabilistic networks (Wellman, 1990a) provided a purely qualitative abstraction of Bayesian networks, using the notion of positive and negative influences between variables. Wellman shows that in many cases such information is sufficient for optimal decision making without the need for the precise specification of probability values. Work by Adnan Darwiche and Matt Ginsberg (1992) extracts the basic properties of conditioning and evidence combination from probability theory and shows that they can also be applied in logical and default reasoning.

The heart disease treatment system described in the chapter is due to Lucas (1996). Other fielded applications of Bayesian networks include the work at Microsoft on inferring computer user goals from their actions (Horvitz *et al.*, 1998) and on filtering junk email (Sahami *et al.*, 1998), the Electric Power Research Institute's work on monitoring power generators (Morjaria *et al.*, 1995), and NASA's work on displaying time-critical information at Mission Control in Houston (Horvitz and Barry, 1995).

Some important early papers on uncertain reasoning methods in AI are collected in the anthologies *Readings in Uncertain Reasoning* (Shafer and Pearl, 1990) and *Uncertainty in Artificial Intelligence* (Kanal and Lemmer, 1986). The most important single publication in the growth of Bayesian networks was undoubtedly the text *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). Several excellent texts, including Lauritzen (1996), Jensen (2001) and Jordan (2003), contain more recent material. New research on probabilistic reasoning appears both in mainstream AI journals such as *Artificial Intelligence* and the *Journal of AI Research,* and in more specialized journals, such as the *International Journal of Approximate Reasoning.* Many papers on graphical models, which include Bayesian networks, appear in statistical journals. The proceedings of the conferences on Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NIPS), and Artificial Intelligence and Statistics (AISTATS) are excellent sources for current research.

EXERCISES

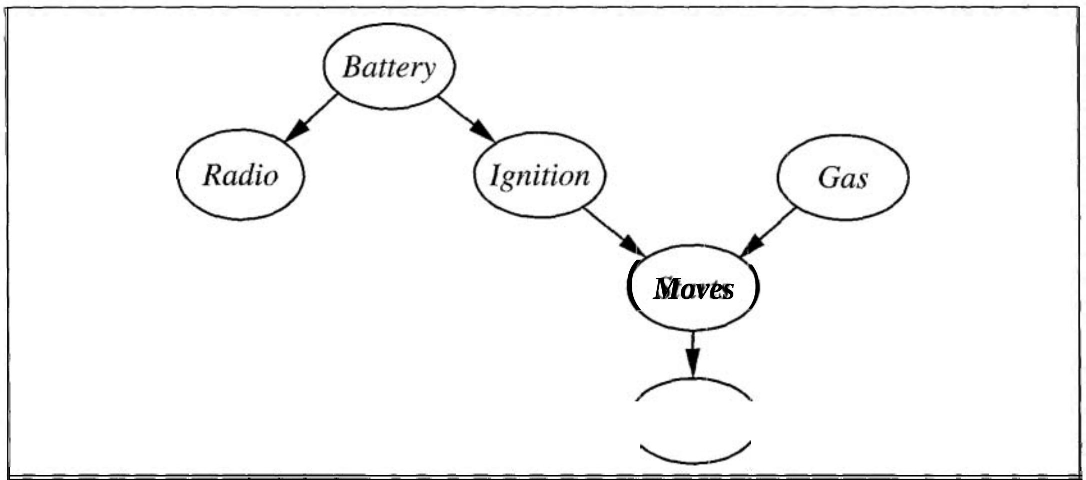


Figure 14.18 A Bayesian network describing some features of a car's electrical system and engine. Each variable is Boolean, and the `true` value indicates that the corresponding aspect of the vehicle is in working order.

14.1 Consider the network for car diagnosis shown in Figure 14.18.

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

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

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

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

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

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

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

b. Is your network a polytree?

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

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

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

14.3 Two astronomers in different parts of the world make measurements M_1 and M_2 of the number of stars N in some small region of the sky, using their telescopes. Normally, there is a small possibility e of error by up to one star in each direction. Each telescope can also (with a much smaller probability f) be badly out of focus (events F_1 and F_2), in which case the scientist will undercount by three or more stars (or, if N is less than 3, fail to detect any stars at all). Consider the three networks shown in Figure 14.19.

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

b. Which is the best network? Explain.

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

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

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

14.4 Consider the network shown in Figure 14.19(ii), and assume that the two telescopes work identically. $N \in \{1,2,3\}$ and $M_1, M_2 \in \{0,1,2,3,4\}$, with the symbolic CPTs as described in Exercise 14.3. Using the enumeration algorithm, calculate the probability distribution $\mathbf{P}(N|M_1 = 2, M_2 = 2)$.

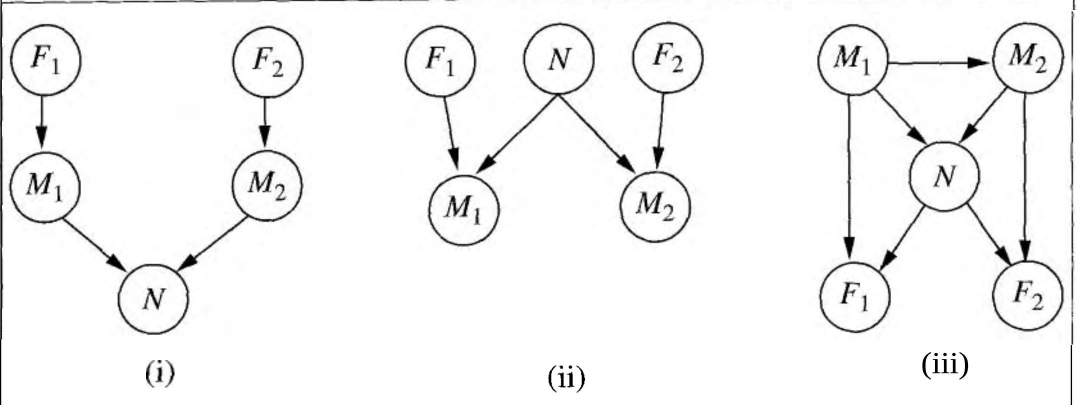


Figure 14.19 Three possible networks for the telescope problem.

14.5 Consider the family of linear Gaussian networks, as illustrated on page 502.

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

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

14.6 The probit distribution defined on page 503 describes the probability distribution for a Boolean child, given a single continuous parent.

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

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

14.7 This exercise is concerned with the variable elimination algorithm in Figure 14.10.

a. Section 14.4 applies variable elimination to the query

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

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

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

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

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

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

a. Prove that any 3-SAT problem can be reduced to exact inference in a Bayesian network constructed to represent the particular problem and hence that exact inference is NP-hard. [*Hint*: Consider a network with one variable for each proposition symbol, one for each clause, and one for the conjunction of clauses.]

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

14.9 Consider the problem of generating a random sample from a specified distribution on a single variable. You can assume that a random number generator is available that returns a random number uniformly distributed between 0 and 1.

CUMULATIVE DISTRIBUTION

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

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

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

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

14.10 The **Markov blanket** of a variable is defined on page 499.

a. Prove that a variable is independent of all other variables in the network, given its Markov blanket.

b. Derive Equation (14.11).

14.11 Consider the query $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$ in Figure 14.11(a) and how MCMC can answer it.

a. How many states does the Markov chain have?

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

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

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

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

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

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

b. Construct the equivalent Bayesian network.

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

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

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