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

14.2 THE SEMANTICS OF BAYESIAN NETWORKS

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

14.2.1 Representing the full joint distribution

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

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

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

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

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

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

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

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

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

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

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

A method for constructing Bayesian networks

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

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

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

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

CHAIN RULE

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

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

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

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

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

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

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

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

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

Compactness and node ordering

LOCALLY STRUCTURED
SPARSE

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

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

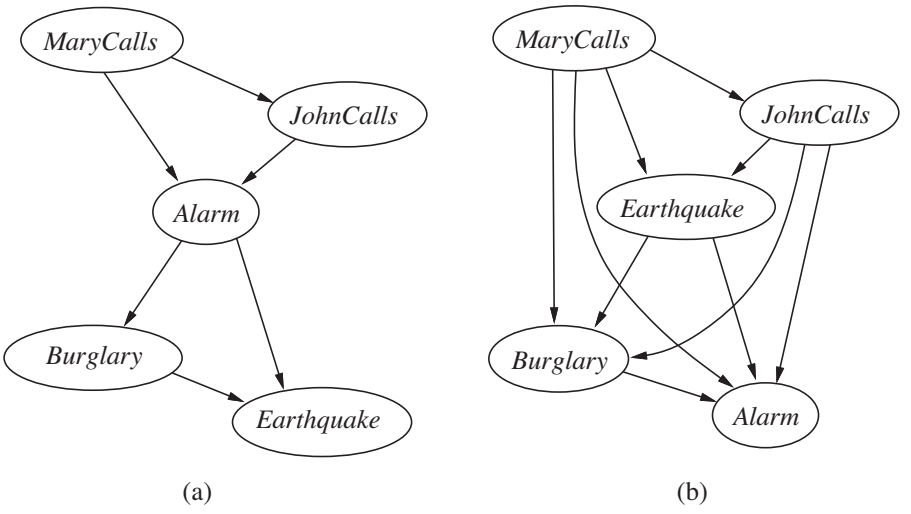

Figure 14.3 Network structure depends on order of introduction. In each network, we have introduced nodes in top-to-bottom order.

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

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

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

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

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

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

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

14.2.2 Conditional independence relations in Bayesian networks

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

DESCENDANT

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

MARKOV BLANKET

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

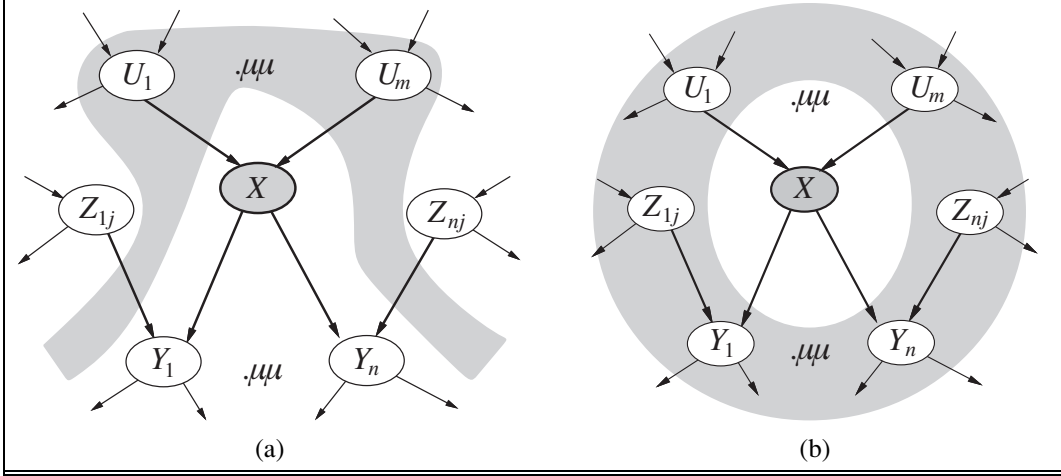

Figure 14.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the gray area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the gray area).

14.3 Efficient Representation of Conditional Distributions

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

CANONICAL DISTRIBUTION

DETERMINISTIC NODES

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

NOISY-OR

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

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

LEAK NODE

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

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

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

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

$Cold$	Flu	$Malaria$	$P(Fever)$	$P(\neg Fever)$
F	F	F	0.0	1.0
F	F	T	0.9	**0.1**
F	T	F	0.8	**0.2**
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	**0.6**
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

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

Bayesian nets with continuous variables

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

DISCRETIZATION

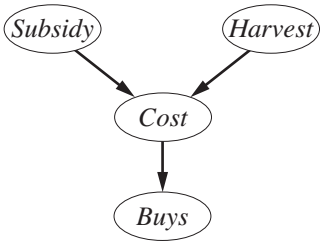

Figure 14.5 A simple network with discrete variables ($Subsidy$ and $Buys$) and continuous variables ($Harvest$ and $Cost$).

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

PARAMETER

NONPARAMETRIC

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

HYBRID BAYESIAN NETWORK

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

LINEAR GAUSSIAN

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

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

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

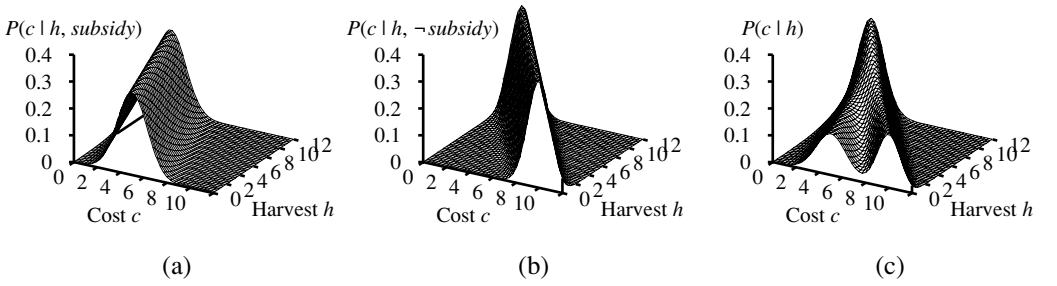

Figure 14.6 The graphs in (a) and (b) show the probability distribution over $Cost$ as a function of $Harvest$ size, with $Subsidy$ true and false, respectively. Graph (c) shows the distribution $P(Cost \mid Harvest)$, obtained by summing over the two subsidy cases.

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

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

CONDITIONAL GAUSSIAN

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

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

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

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

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

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

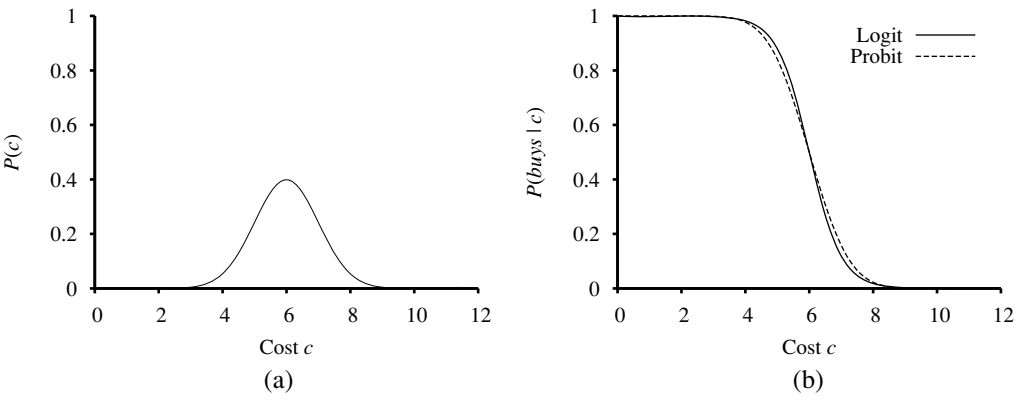

Figure 14.7 (a) A normal (Gaussian) distribution for the cost threshold, centered on $\mu = 6.0$ with standard deviation $\sigma = 1.0$. (b) Logit and probit distributions for the probability of $buys$ given $cost$, for the parameters $\mu = 6.0$ and $\sigma = 1.0$.

PROBIT DISTRIBUTION

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

LOGIT DISTRIBUTION

LOGISTIC FUNCTION

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

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

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

14.4 EXACT INFERENCE IN BAYESIAN NETWORKS

EVENT

HIDDEN VARIABLE

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

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

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

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

14.4.1 Inference by enumeration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

14.4.2 The variable elimination algorithm

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

VARIABLE ELIMINATION

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

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

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

FACTOR

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

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

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

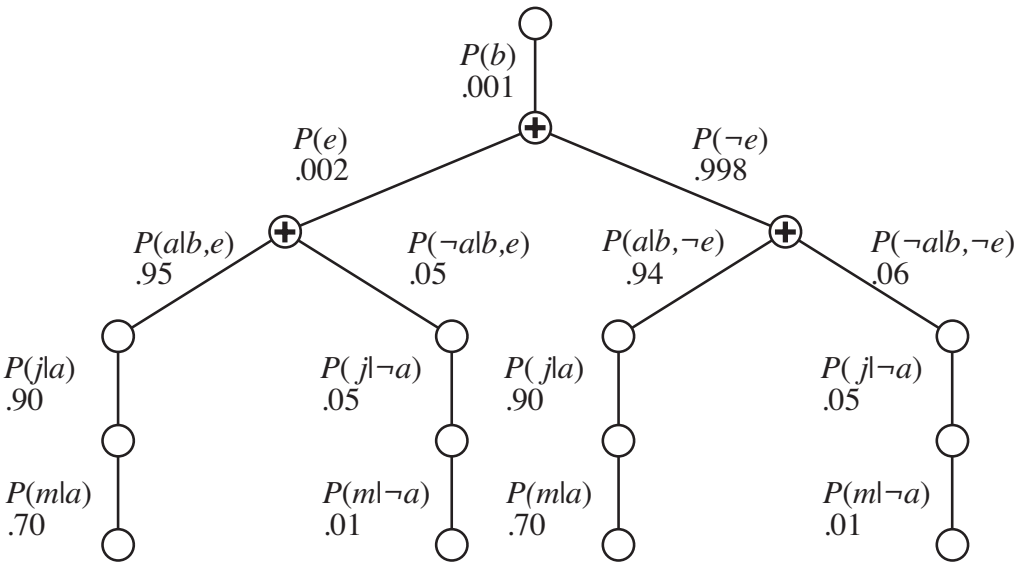

Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

function ENUMERATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayes net with variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ /* **Y** = *hidden variables* */

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

---

function ENUMERATE-ALL($vars$, **e**) **returns** a real number
  if EMPTY?($vars$) **then return** 1.0
  $Y \leftarrow$ FIRST($vars$)
  if Y has value y in **e**
    then return $P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), **e**)
    else return $\sum_y P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_y$)
      where $\mathbf{e}_y$ is **e** extended with $Y = y$

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

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

POINTWISE PRODUCT

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

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

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

  Now we are left with the expression

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

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

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

  This leaves the expression

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

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

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

Operations on factors

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

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

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

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

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

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

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

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

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

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

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

Variable ordering and variable relevance

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

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

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

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

```
function ELIMINATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayesian network specifying joint distribution P(X_1, ..., X_n)

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

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

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

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

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

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

14.4.3 The complexity of exact inference

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

SINGLY CONNECTED

POLYTREE

MULTIPLY CONNECTED

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

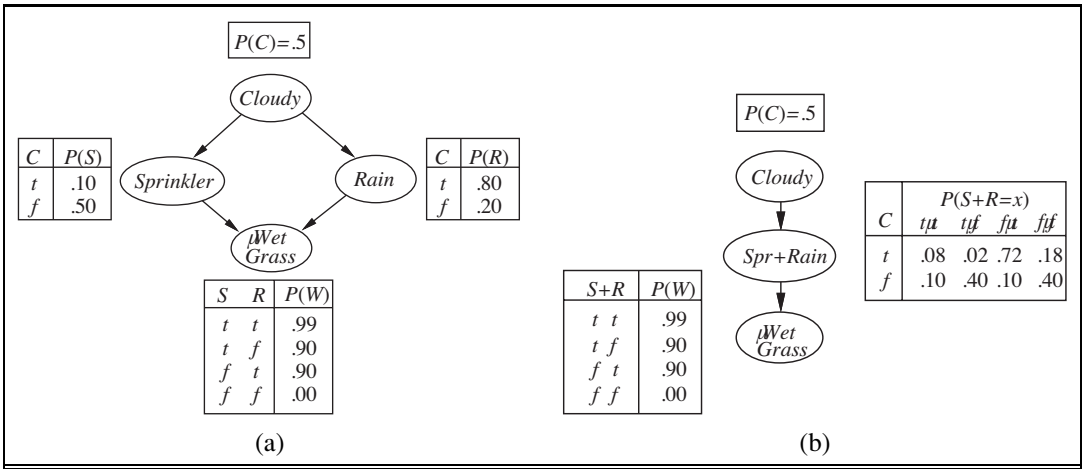

Figure 14.12 (a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

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

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

14.4.4 Clustering algorithms

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

CLUSTERING

JOIN TREE

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

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

14.5 Approximate Inference in Bayesian Networks

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

MONTE CARLO

14.5.1 Direct sampling methods

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

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

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

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