CHAPTER 13

PROBABILISTIC REASONING

In which we explain how to build efficient network models to reason under uncertainty according to the laws of probability theory, and how to distinguish between correlation and causality.

Chapter 12 introduced the basic elements of probability theory and noted the importance of independence and conditional independence relationships in simplifying probabilistic representations of the world. This chapter introduces a systematic way to represent such relationships explicitly in the form of **Bayesian networks**. We define the syntax and semantics of these networks and show how they can be used to capture uncertain knowledge in a natural and efficient way. We then show how probabilistic inference, although computationally intractable in the worst case, can be done efficiently in many practical situations. We also describe a variety of approximate inference algorithms that are often applicable when exact inference is infeasible. Chapter 18 extends the basic ideas of Bayesian networks to more expressive formal languages for defining probability models.

13.1 Representing Knowledge in an Uncertain Domain

In Chapter 12, we saw that the full joint probability distribution can answer any question about the domain, but can become intractably large as the number of variables grows. Furthermore, specifying probabilities for possible worlds one by one is unnatural and tedious.

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

Bayesian network

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

1. Each node corresponds to a random variable, which may be discrete or continuous.
2. Directed links or arrows connect pairs of nodes. If there is an arrow from node X to node Y, X is said to be a *parent* of Y. The graph has no directed cycles and hence is a directed acyclic graph, or DAG.
3. Each node X_i has associated probability information $\theta(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node using a finite number of **parameters**.

Parameter

[1] Bayesian networks, often abbreviated to "Bayes net," were called **belief networks** in the 1980s and 1990s. A **causal network** is a Bayes net with additional constraints on the meaning of the arrows (see Section 13.5). The term **graphical model** refers to a broader class that includes Bayesian networks.

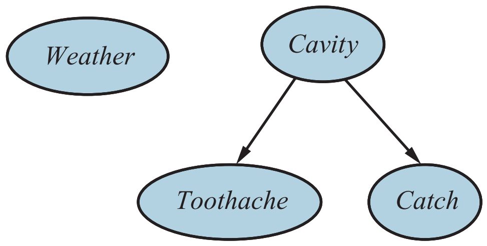


Figure 13.1 A simple Bayesian network in which *Weather* is independent of the other three variables and *Toothache* and *Catch* are conditionally independent, given *Cavity*.

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

Recall the simple world described in Chapter 12, consisting of the variables *Toothache*, *Cavity*, *Catch*, and *Weather*. We argued that *Weather* is independent of the other variables; furthermore, we argued that *Toothache* and *Catch* are conditionally independent, given *Cavity*. These relationships are represented by the Bayes net structure shown in Figure 13.1. Formally, the conditional independence of *Toothache* and *Catch*, given *Cavity*, is indicated by the *absence* of a link between *Toothache* and *Catch*. Intuitively, the network represents the fact that *Cavity* is a direct cause of *Toothache* and *Catch*, whereas no direct causal relationship exists between *Toothache* and *Catch*.

Now consider the following example, which is just a little more complex. You have a new burglar alarm installed at home. It is fairly reliable at detecting a burglary, but is occasionally set off by minor earthquakes. (This example is due to Judea Pearl, a resident of earthquake-prone Los Angeles.) You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm. John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and often misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.

A Bayes net for this domain appears in Figure 13.2. The network structure shows that burglary and earthquakes directly affect the probability of the alarm's going off, but whether John and Mary call depends only on the alarm. The network thus represents our assumptions that they do not perceive burglaries directly, they do not notice minor earthquakes, and they do not confer before calling.

The local probability information attached to each node in Figure 13.2 takes the form of a **conditional probability table (CPT)**. (CPTs can be used only for discrete variables; other representations, including those suitable for continuous variables, are described in Sec-

Conditional probability table (CPT)

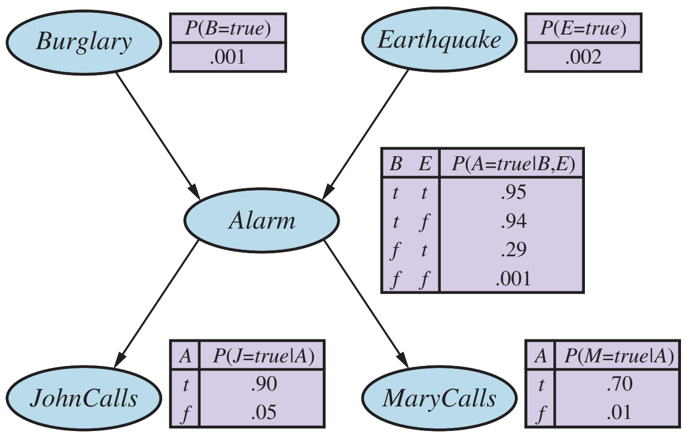


Figure 13.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

Conditioning case

tion 13.2.) Each row in a CPT contains the conditional probability of each node value for a **conditioning case**. A conditioning case is just a possible combination of values for the parent nodes—a miniature possible world, if you like. Each row must sum to 1, because the entries represent an exhaustive set of cases for the variable. For Boolean variables, once you know that the probability of a true value is p, the probability of false must be $1-p$, so we often omit the second number, as in Figure 13.2. In general, a table for a Boolean variable with k Boolean parents contains 2^k independently specifiable probabilities. A node with no parents has only one row, representing the prior probabilities of each possible value of the variable.

Notice that the network does not have nodes corresponding to Mary's currently listening to loud music or to the telephone ringing and confusing John. These factors are summarized in the uncertainty associated with the links from *Alarm* to *JohnCalls* and *MaryCalls*. This shows both laziness and ignorance in operation, as explained on page 404: 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.

13.2 The Semantics of Bayesian Networks

The *syntax* of a Bayes net consists of a directed acyclic graph with some local probability information attached to each node. The *semantics* defines how the syntax corresponds to a joint distribution over the variables of the network.

Assume that the Bayes net contains n variables, $X_1,\ldots,X_n$. A generic entry in the joint distribution is then $P(X_1=x_1 \land \ldots \land X_n=x_n)$, or $P(x_1,\ldots,x_n)$ for short. The semantics of

Bayes nets defines each entry in the joint distribution as follows:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} \theta(x_i \,|\, parents(X_i))\,, \tag{13.1}$$

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 local conditional distributions in the Bayes net.

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 simply multiply the relevant entries from the local conditional distributions (abbreviating the variable names):

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

Section 12.3 explained that the full joint distribution can be used to answer any query about the domain. If a Bayes net is a representation of the joint distribution, then it too can be used to answer any query, by summing all the relevant joint probability values, each calculated by multiplying probabilities from the local conditional distributions. Section 13.3 explains this in more detail, but also describes methods that are much more efficient.

So far, we have glossed over one important point: what is the meaning of the numbers that go into the local conditional distributions $\theta(x_i \,|\, parents(X_i))$? It turns out that from Equation (13.1) we can prove that the parameters $\theta(x_i \,|\, parents(X_i))$ are exactly the conditional probabilities $P(x_i \,|\, parents(X_i))$ implied by the joint distribution. Remember that the conditional probabilities can be computed from the joint distribution as follows:

$$\begin{aligned} P(x_i \,|\, parents(X_i)) &\equiv \frac{P(x_i, parents(X_i))}{P(parents(X_i))} \\ &= \frac{\sum_{\mathbf{y}} P(x_i, parents(X_i), \mathbf{y})}{\sum x'_i, \mathbf{y} P(x'_i, parents(X_i), \mathbf{y})} \end{aligned}$$

where $\mathbf{y}$ represents the values of all variables other than X_i and its parents. From this last line one can prove that $P(x_i \,|\, parents(X_i)) = \theta(x_i \,|\, parents(X_i))$ (Exercise 13.CPTE). Hence, we can rewrite Equation (13.1) as

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \,|\, parents(X_i))\,. \tag{13.2}$$

This means that when one estimates values for the local conditional distributions, they need to be the actual conditional probabilities for the variable given its parents. So, for example, when we specify $\theta(JohnCalls = true \,|\, Alarm = true) = 0.90$, it should be the case that about 90% of the time when the alarm sounds, John calls. The fact that each parameter of the network has a precise meaning in terms of only a small set of variables is crucially important for robustness and ease of specification of the models.

A method for constructing Bayesian networks

Equation (13.2) defines what a given Bayes net 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 (13.2) 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 408):

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

Then we repeat the process, reducing each joint probability to a conditional probability and a joint probability on a smaller set of variables. 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 is called the **chain rule**. It holds for any set of random variables. Comparing it with Equation (13.2), we see that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i in the network,

$$\mathbf{P}(X_i \,|\, X_{i-1},\ldots,X_1) = \mathbf{P}(X_i \,|\, Parents(X_i))\,, \tag{13.3}$$

Topological ordering

provided that $Parents(X_i) \subseteq \{X_{i-1},\ldots,X_1\}$. This last condition is satisfied by numbering the nodes in **topological order**—that is, in any order consistent with the directed graph structure. For example, the nodes in Figure 13.2 could be ordered B,E,A,J,M; E,B,A,M,J; and so on.

What Equation (13.3) 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 a minimal set of parents for X_i from $X_1,\ldots,X_{i-1}$, such that Equation (13.3) 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 13.2 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:

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

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 Bayes nets is that they contain no redundant probability values. If there is no redundancy, then there is no chance for inconsistency: *it is impossible for the knowledge engineer or domain expert to create a Bayesian network that violates the axioms of probability.*

Compactness and node ordering

As well as being a complete and nonredundant representation of the domain, a Bayes net 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.

Locally structured

Sparse

In the case of Bayes nets, 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 $2^k \cdot n$ 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.

Specifying the conditional probability tables for a fully connected network, in which each variable has all of its predecessors as parents, requires the same amount of information as specifying the joint distribution in tabular form. For this reason, we often leave out links even though a slight dependency exists, because the slight gain in accuracy is not worth the the additional complexity in the network. For example, one might object to our burglary network on the grounds that if there is a large 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 the importance of getting more accurate probabilities compared with the cost of specifying the extra information.

Even in a locally structured domain, we will get a compact Bayes net 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 13.3(a). The process goes as follows:

- Adding *MaryCalls*: No parents.
- Adding *JohnCalls*: If Mary calls, that probably means the alarm has gone off, which makes 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.

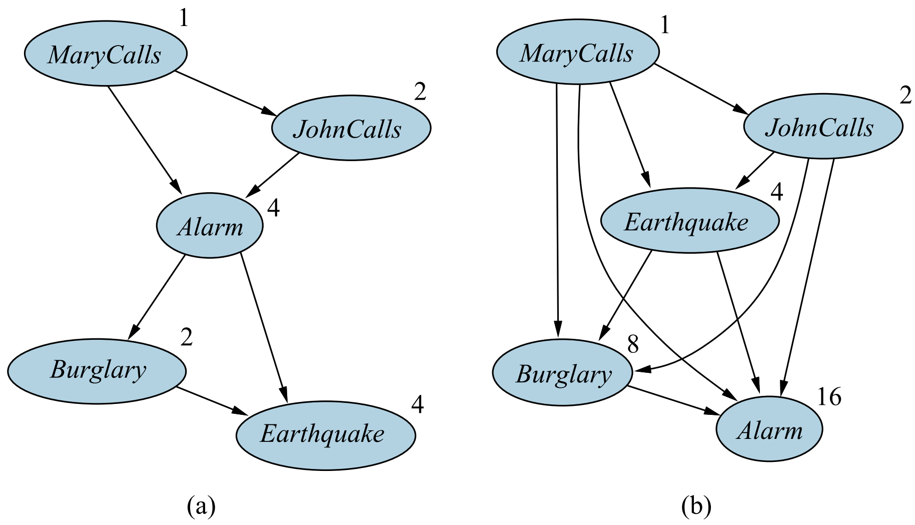


Figure 13.3 Network structure and number of parameters depends on order of introduction. (a) The structure obtained with ordering M,J,A,B,E. (b) The structure obtained with M,J,E,B,A. Each node is annotated with the number of parameters required; 13 in all for (a) and 31 for (b). In Figure 13.2, only 10 parameters were required.

The resulting network has two more links than the original network in Figure 13.2 and requires 13 conditional probabilities rather than 10. What's worse, some of the links represent tenuous relationships that require difficult and unnatural probability judgments, such as assessing the probability of *Earthquake*, given *Burglary* and *Alarm*. This phenomenon is quite general and is related to the distinction between **causal** and **diagnostic** models introduced in Section 12.5.1 (see also Exercise 13.WUMD). *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.* For example, in the domain of medicine, 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. Section 13.5 explores the idea of causal models in more depth.

Figure 13.3(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 two versions in Figure 13.3 simply fail to represent all the conditional independence relationships and hence end up specifying a lot of unnecessary numbers instead.

13.2.1 Conditional independence relations in Bayesian networks

From the semantics of Bayes nets as defined in Equation (13.2), we can derive a number of conditional independence properties. We have already seen the property that a variable is conditionally independent of its other predecessors, given its parents. It is also possible to prove the more general "non-descendants" property that:

Descendant

*Each variable is conditionally independent of its non-**descendants**, given its parents.*

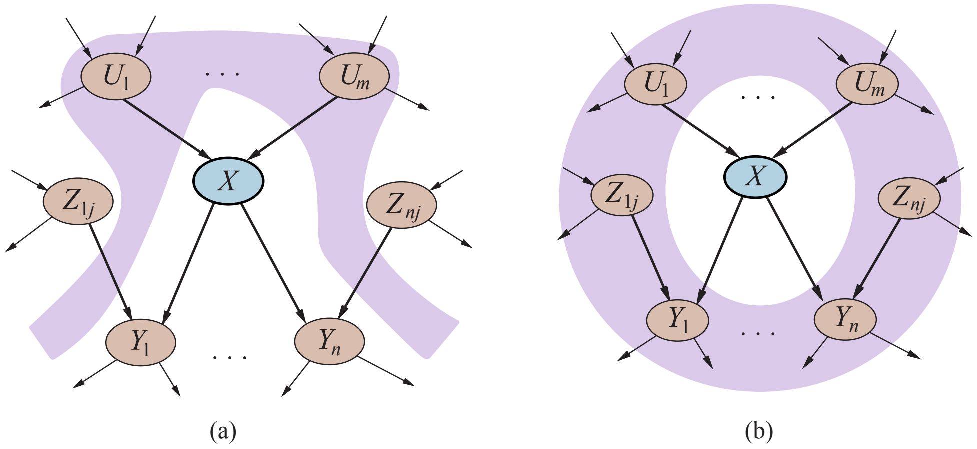


Figure 13.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the lavender area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the lavender area).

For example, in Figure 13.2, the variable *JohnCalls* is independent of *Burglary*, *Earthquake*, and *MaryCalls* given the value of *Alarm*. The definition is illustrated in Figure 13.4(a).

It turns out that the non-descendants property combined with interpretation of the network parameters $\theta(X_i \mid Parents(X_i))$ as conditional probabilities $\mathbf{P}(X_i \mid Parents(X_i))$ suffices to reconstruct the full joint distribution given in Equation (13.2). In other words, one can view the semantics of Bayes nets in a different way: instead of defining the full joint distribution as the product of conditional distributions, the network defines a set of conditional independence properties. The full joint distribution can be derived from those properties.

Another important independence property is implied by the non-descendants property:

> a variable is conditionally independent of all other nodes in the network, given its parents, children, and children's parents—that is, given its **Markov blanket**.

Markov blanket

(Exercise 13.MARB asks you to prove this.) For example, the variable *Burglary* is independent of *JohnCalls* and *MaryCalls*, given *Alarm* and *Earthquake*. This property is illustrated in Figure 13.4(b). The Markov blanket property makes possible inference algorithms that use completely local and distributed stochastic sampling processes, as explained in Section 13.4.2.

The most general conditional independence question one might ask in a Bayes net is whether a set of nodes **X** is conditionally independent of another set **Y**, given a third set **Z**. This can be determined efficiently by examining the Bayes net to see whether **Z d-separates X** and **Y**. The process works as follows:

D-separation

1. Consider just the **ancestral subgraph** consisting of **X**, **Y**, **Z**, and their ancestors.
2. Add links between any unlinked pair of nodes that share a common child; now we have the so-called **moral graph**.
3. Replace all directed links by undirected links.
4. If **Z** blocks all paths between **X** and **Y** in the resulting graph, then **Z** d-separates **X** and **Y**. In that case, **X** is conditionally independent of **Y**, given **Z**. Otherwise, the original Bayes net does not require conditional independence.

Ancestral subgraph

Moral graph

In brief, then, d-separation means separation in the undirected, moralized, ancestral subgraph. Applying the definition to the burglary network in Figure 13.2, we can deduce that *Burglary* and *Earthquake* are independent given the empty set (i.e., they are absolutely independent); that they are *not* necessarily conditionally independent given *Alarm*; and that *JohnCalls* and *MaryCalls* are conditionally independent given *Alarm*. Notice also that the Markov blanket property follows directly from the d-separation property, since a variable's Markov blanket d-separates it from all other variables.

13.2.2 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 just by naming the pattern and perhaps supplying a few 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, the *BestPrice* for a car is the minimum of the prices at each dealer in the area; and the *WaterStored* in a reservoir at year's end is the sum of the original amount, plus the inflows (rivers, runoff, precipitation) and minus the outflows (releases, evaporation, seepage).

Many Bayes net systems allow the user to specify deterministic functions using a general-purpose programming language; this makes it possible to include complex elements such as global climate models or power-grid simulators within a probabilistic model.

Context-specific independence

Another important pattern that occurs often in practice is **context-specific independence** or CSI. A conditional distribution exhibits CSI if a variable is conditionally independent of some of its parents given *certain values* of others. For example, let's suppose that the *Damage* to your car occurring during a given period of time depends on the *Ruggedness* of your car and whether or not an *Accident* occurred in that period. Clearly, if *Accident* is false, then the *Damage*, if any, doesn't depend on the *Ruggedness* of your car. (There might be vandalism damage to the car's paintwork or windows, but we'll assume all cars are equally subject to such damage.) We say that *Damage* is context-specifically independent of *Ruggedness* given *Accident*=*false*. Bayes net systems often implement CSI using an if-then-else syntax for specifying conditional distributions; for example, one might write

$$\mathbf{P}(Damage \mid Ruggedness, Accident) = \\ \quad \textbf{if } (Accident = false) \textbf{ then } d_1 \textbf{ else } d_2(Ruggedness)$$

where d_1 and d_2 represent arbitrary distributions. As with determinism, the presence of CSI in a network may facilitate efficient inference. All of the exact inference algorithms mentioned in Section 13.3 can be modified to take advantage of CSI to speed up computation.

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*

Cold	*Flu*	*Malaria*	$P(fever \mid \cdot)$	$P(\neg fever \mid \cdot)$
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	$0.012 = 0.6 \times 0.2 \times 0.1$

Figure 13.5 A complete conditional probability table for $\mathbf{P}(Fever \mid Cold, Flu, Malaria)$, assuming a noisy-OR model with the the three q-values shown in bold.

are 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_j for each parent. Let us suppose these individual inhibition probabilities are as follows:

Leak node

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

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. Figure 13.5 illustrates this calculation.

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 parameters instead of 133,931,430 for a network with full CPTs.

13.2.3 Bayesian nets with continuous variables

Many real-world problems involve continuous quantities, such as height, mass, temperature, and money. By definition, continuous variables have an infinite number of possible values, so it is impossible to specify conditional probabilities explicitly for each value. One way to handle continuous variables is with **discretization**—that is, dividing up the possible values into a fixed set of intervals. For example, temperatures could be divided into three categories: (<0°C), (0°C−100°C), and (>100°C). In choosing the number of categories, there is a tradeoff between loss of accuracy and large CPTs which can lead to slow run times.

Discretization

Another approach is to define a continuous variable using one of the standard families of probability density functions (see Appendix A). For example, a Gaussian (or normal) distribution $\mathcal{N}(x;\mu,\sigma^2)$ is specified by just two parameters, the mean μ and the variance σ^2. 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 19.

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 13.6, 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 $\mathbf{P}(Cost \mid Harvest, subsidy)$ and $\mathbf{P}(Cost \mid Harvest, \neg subsidy)$. 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** conditional distribution, in which the child has a Gaussian distribution whose mean μ varies linearly with the value of the parent and whose standard deviation σ is fixed. We need two distributions, one for *subsidy* and one for $\neg$*subsidy*, with different parameters:

Linear–Gaussian

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

For this example, then, the conditional distribution for *Cost* is specified by naming the linear–Gaussian distribution and providing the parameters a_t, b_t, σ_t, a_f, b_f, and σ_f. Figures 13.7(a) and (b) show these two relationships. Notice that in each case the slope of c versus h is negative, because cost decreases as the harvest size 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 13.7(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.

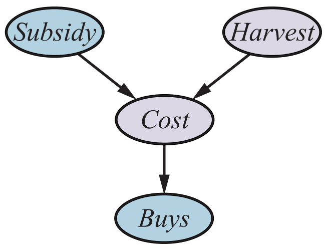
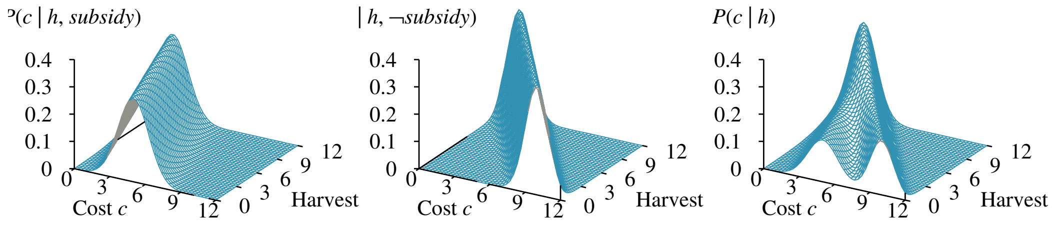


Figure 13.6 A simple network with discrete variables (*Subsidy* and *Buys*) and continuous variables (*Harvest* and *Cost*).



(a) (b) (c)

Figure 13.7 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.

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 13.LGEX). Furthermore, the posterior distribution given any evidence also has this property.[2] 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 13.6. 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:

$$\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(s;0,1)ds\,.$$

[2] 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 13.3, we see that inference for networks of discrete variables is NP-hard.

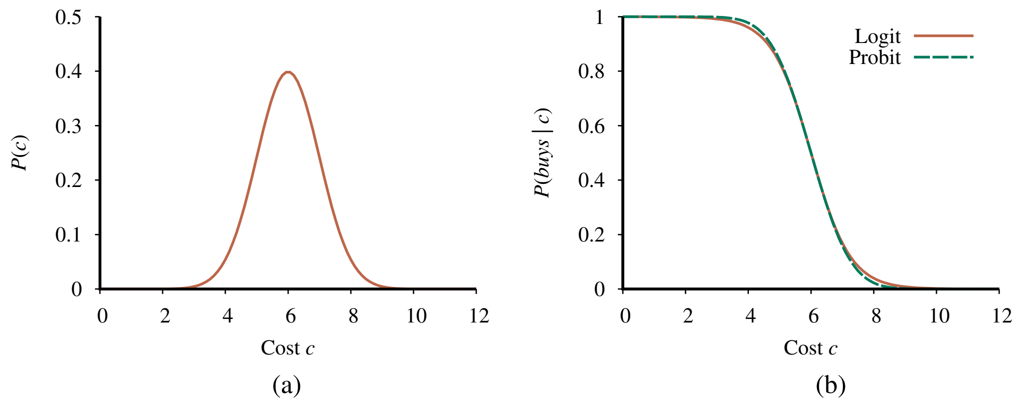


Figure 13.8 (a) A normal (Gaussian) distribution for the cost threshold, centered on $\mu = 6.0$ with standard deviation $\sigma = 1.0$. (b) Expit and probit models for the probability of *buys* given *cost*, for the parameters $\mu = 6.0$ and $\sigma = 1.0$.

$\Phi(x)$ is an increasing function of x, whereas the probability of buying decreases with cost, so here we flip the function around:

$$P(buys \mid Cost = c) = 1 - \Phi((c - \mu)/\sigma)\,,$$

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** model (pronounced "pro-bit" and short for "probability unit") is illustrated in Figure 13.8(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.

Probit

An alternative to the probit model is the **expit** or **inverse logit** model. It uses the **logistic function** $1/(1 + e^{-x})$ to produce a soft threshold—it maps any x to a value between 0 and 1. Again, for our example, we flip it around to make a decreasing function; we also scale the exponent by $4/\sqrt{2\pi}$ to match the probit's slope at the mean:

Expit
Inverse logit
Logistic function

$$P(buys \mid Cost = c) = 1 - \frac{1}{1 + exp(-\frac{4}{\sqrt{2\pi}} \cdot \frac{c-\mu}{\sigma})}\,.$$

This is illustrated in Figure 13.8(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 logistic function is sometimes easier to deal with mathematically. It is used widely in machine learning. Both models can be generalized to handle multiple continuous parents by taking a linear combination of the parent values. This also works for discrete parents if their values are integers; for example, with k Boolean parents, each viewed as having values 0 or 1, the input to the expit or probit distribution would be a weighted linear combination with k parameters, yielding a model quite similar to the noisy-OR model discussed earlier.

13.2.4 Case study: Car insurance

A car insurance company receives an application from an individual to insure a specific vehicle and must decide on the appropriate annual premium to charge, based on the anticipated claims it will pay out for this applicant. The task is to build a Bayes net that captures the

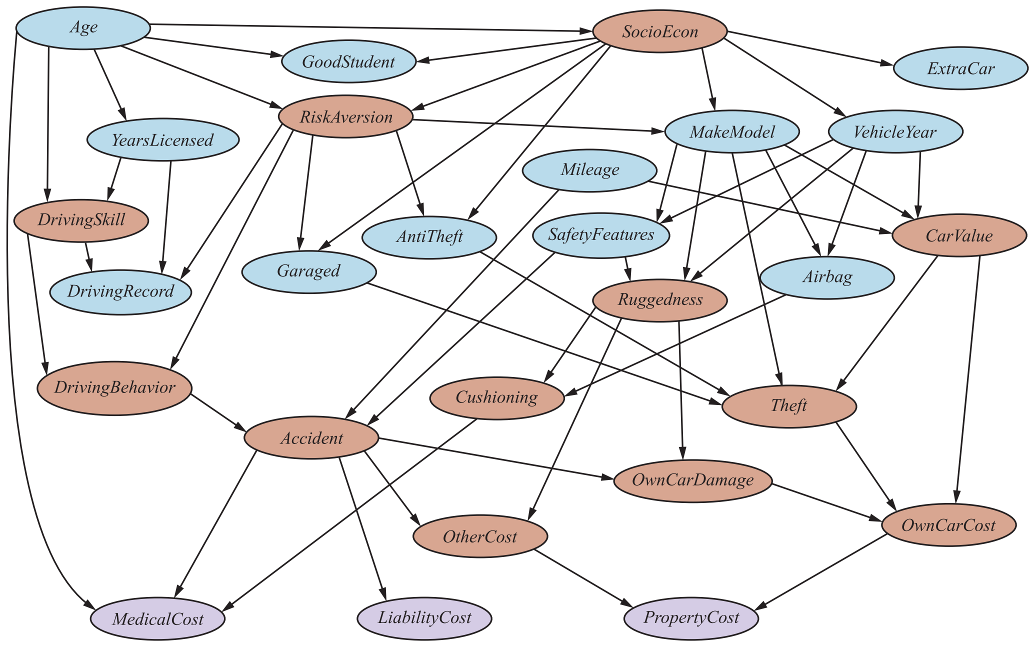


Figure 13.9 A Bayesian network for evaluating car insurance applications.

causal structure of the domain and gives an accurate, well-calibrated distribution over the output variables given the evidence available from the application form.[3] The Bayes net will include **hidden variables** that are neither input nor output variables, but are essential for structuring the network so that it is reasonably sparse with a manageable number of parameters. The hidden variables are shaded brown in Figure 13.9.

Hidden variable

The claims to be paid out—shaded lavender in Figure 13.9—are of three kinds: the *MedicalCost* for any injuries sustained by the applicant; the *LiabilityCost* for lawsuits filed by other parties against the applicant and the company; and the *PropertyCost* for vehicle damage to either party and vehicle loss by theft. The application form asks for the following input information (the light blue nodes in Figure 13.9):

- About the applicant: *Age*; *YearsLicensed*—how long since a driving license was first obtained; *DrivingRecord*—some summary, perhaps based on "points," of recent accidents and traffic violations; and (for students) a *GoodStudent* indicator for a grade-point average of 3.0 (B) on a 4-point scale.
- About the vehicle: the *MakeModel* and *VehicleYear*; whether it has an *Airbag*; and some summary of *SafetyFeatures* such as anti-lock braking and collision warning.
- About the driving situation: the annual *Mileage* driven and how securely the vehicle is *Garaged*, if at all.

[3] The network shown in Figure 13.9 is not in actual use, but its structure has been vetted with insurance experts. In practice, the information requested on application forms varies by company and jurisdiction—for example, some ask for *Gender*—and the model could certainly be made more detailed and sophisticated.

Now we need to think about how to arrange these into a causal structure. The key hidden variables are whether or not a *Theft* or *Accident* will occur in the next time period. Obviously, one cannot ask the applicant to predict these; they have to be inferred from the available information and the insurer's previous experience.

What are the causal factors leading to *Theft*? The *MakeModel* is certainly important—some models are stolen much more often than others because there is an efficient resale market for vehicles and parts; the *CarValue* also matters, because an old, beat-up, or high-mileage vehicle has lower resale value. Moreover, a vehicle that is *Garaged* and has an *AntiTheft* device is harder to steal. The hidden variable *CarValue* depends in turn on the *MakeModel*, *VehicleYear*, and *Mileage*. *CarValue* also dictates the loss amount when a *Theft* occurs, so that is one of the contributors to *OwnCarCost* (the other being accidents, which we will get to shortly).

It is common in models of this type to introduce another hidden variable, *SocioEcon*, the socioeconomic category of the applicant. This is thought to influence a wide range of behaviors and characteristics. In our model, there is no *direct* evidence in the form of observed income and occupation variables;[4] but *SocioEcon* influences *MakeModel* and *VehicleYear*; it also affects *ExtraCar* and *GoodStudent*, and depends somewhat on *Age*.

For any insurance company, perhaps the most important hidden variable is *RiskAversion*: people who are risk-averse are good insurance risks! *Age* and *SocioEcon* affect *RiskAversion*, and its "symptoms" include the applicant's choice of whether the vehicle is *Garaged* and has *AntiTheft* devices and *SafetyFeatures*.

In predicting future accidents, the key is the applicant's future *DrivingBehavior*, which is influenced by both *RiskAversion* and *DrivingSkill*; the latter in turn depends on *Age* and *YearsLicensed*. The applicant's past driving behavior is reflected in the *DrivingRecord*, which also depends on *RiskAversion* and *DrivingSkill* as well as on *YearsLicensed* (because someone who started driving only recently may not have had time to accumulate a litany of accidents and violations). In this way, *DrivingRecord* provides evidence about *RiskAversion* and *DrivingSkill*, which in turn help to predict future *DrivingBehavior*.

We can think of *DrivingBehavior* as a per-mile tendency to drive in an accident-prone way; whether an *Accident* actually occurs in a fixed time period depends also on the annual *Mileage* and on the *SafetyFeatures* of the vehicle. If an *Accident* occurs, there are three kinds of costs: the *MedicalCost* for the applicant depends on *Age* and *Cushioning*, which depends in turn on the *Ruggedness* of the car and whether it has an *Airbag*; the *LiabilityCost* (medical, pain and suffering, loss of income, etc.) for the other driver; and the *PropertyCost* for the applicant and the other driver, both of which depend (in different ways) on the car's *Ruggedness* and on the applicant's *CarValue*.

We have illustrated the kind of reasoning that goes into developing the topology and hidden variables in a Bayes net. We also need to specify the ranges and the conditional distributions for each variable. For the ranges, the primary decision is often whether to make the variable discrete or continuous. For example, the *Ruggedness* of the vehicle could be a continuous variable between 0 and 1, or a discrete variable with range $\{TinCan, Normal, Tank\}$.

[4] Some insurance companies also acquire the applicant's credit history to help in assessing risk; this provides considerably more information about socioeconomic category. Whenever using hidden variables of this kind, one must be careful that they do not inadvertently become proxies for variables such as race that may not be used in insurance decisions. Techniques for avoiding biases of this kind are described in Chapter 19.

Continuous variables provide more precision, but they make exact inference impossible except in a few special cases. A discrete variable with many possible values can make it tedious to fill in the correspondingly large conditional probability tables and makes exact inference more expensive unless the variable's value is always observed. For example, *MakeModel* in a real system would have thousands of possible values, and this causes its child *CarValue* to have an enormous CPT that would have to be filled in from industry databases; but, because the *MakeModel* is always observed, this does not contribute to inference complexity: in fact, the observed values for the three parents pick out exactly one relevant row of the CPT for *CarValue*.

The conditional distributions in the model are given in the code repository for the book; we provide a version with only discrete variables, for which exact inference can be performed. In practice, many of the variables would be continuous and the conditional distributions would be learned from historical data on applicants and their insurance claims. We will see how to learn Bayes net models from data in Chapter 21.

The final question is, of course, how to do inference in the network to make predictions. We turn now to this question. For each inference method that we describe, we will evaluate the method on the insurance net to measure the time and space requirements of the method.

13.3 Exact Inference in Bayesian Networks

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**—usually, some assignment of values to a set of **evidence variables**.[5] 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. (For example, we can solve the query $\mathbf{P}(U,V\,|\,\mathbf{e})$ by multiplying $\mathbf{P}(V\,|\,\mathbf{e})$ and $\mathbf{P}(U\,|\,V,\mathbf{e})$.) We will use the notation from Chapter 12: 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; $\mathbf{Y}$ denotes the hidden (nonevidence, nonquery) variables $Y_1,\ldots,Y_\ell$. Thus, the complete set of variables is $\{X\}\cup\mathbf{E}\cup\mathbf{Y}$. A typical query asks for the posterior probability distribution $\mathbf{P}(X\,|\,\mathbf{e})$.

Event

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 discuss exact algorithms for computing posterior probabilities as well as the complexity of this task. It turns out that the general case is intractable, so Section 13.4 covers methods for approximate inference.

13.3.1 Inference by enumeration

Chapter 12 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 (12.9), which we repeat here for convenience:

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

[5] Another widely studied task is finding the **most probable explanation** for some observed evidence. This and other tasks are discussed in the notes at the end of the chapter.

Now, a Bayes net gives a complete representation of the full joint distribution. More specifically, Equation (13.2) on page 433 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 Bayes net 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 (12.9), using initial letters for the variables to shorten the expressions, we have

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

The semantics of Bayes nets (Equation (13.2)) then gives us an expression in terms of CPT entries. For simplicity, we do this just for $Burglary = true$:

$$P(b \mid j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a)\,. \tag{13.4}$$

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, there will be $O(2^n)$ terms in the sum, each a product of $O(n)$ probability values. A naive implementation would therefore have complexity $O(n2^n)$.

This can be reduced to $O(2^n)$ by taking advantage of the nested structure of the computation. In symbolic terms, this means moving the summations inwards as far as possible in expressions such as Equation (13.4). We can do this because not all the factors in the product of probabilities depend on all the variables. Thus we have

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e)P(j \mid a)P(m \mid a)\,. \tag{13.5}$$

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 as a tree in Figure 13.10. Using the numbers from Figure 13.2, 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 ENUMERATION-ASK algorithm in Figure 13.11 evaluates these expression trees using depth-first, left-to-right recursion. The algorithm is very similar in structure to the backtracking algorithm for solving CSPs (Figure 5.5) and the DPLL algorithm for satisfiability (Figure 7.17). Its space complexity 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 (not counting the evidence variables) is always $O(2^n)$—better than the $O(n2^n)$ for the simple approach described earlier, but still rather grim. For the insurance network in Figure 13.9, which is relatively small, exact inference using enumeration requires around 227 million arithmetic operations for a typical query on the cost variables.

If you look carefully at the tree in Figure 13.10, however, you will see that it contains *repeated subexpressions*. 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 key to efficient inference in Bayes nets is avoiding such wasted computations. The next section describes a general method for doing this.

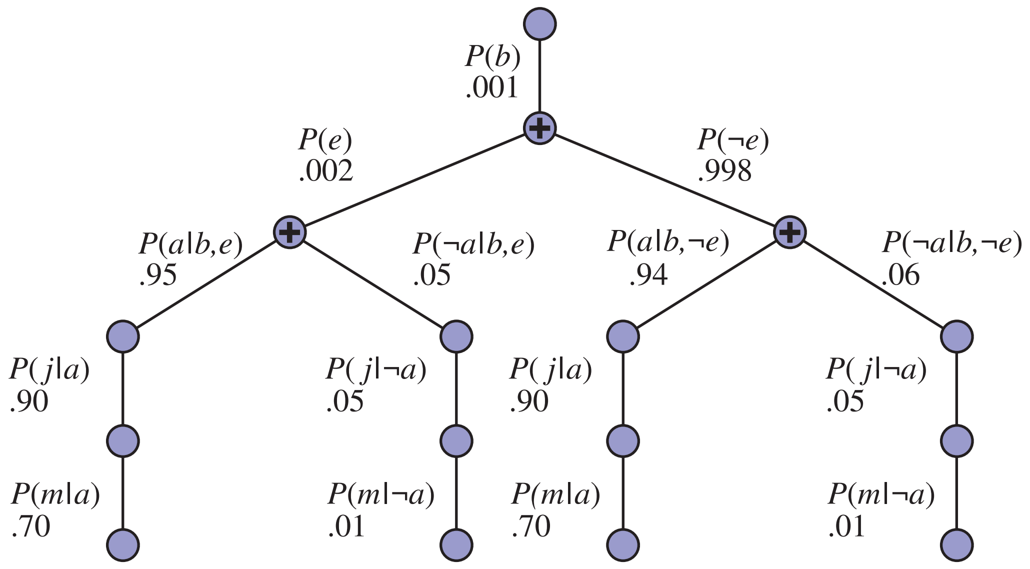


Figure 13.10 The structure of the expression shown in Equation (13.5). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

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

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

function ENUMERATE-ALL($vars$, $\mathbf{e}$) returns a real number
   if EMPTY?($vars$) then return 1.0
   $V \leftarrow$ FIRST($vars$)
   if $V$ is an evidence variable with value $v$ in $\mathbf{e}$
      then return $P(v \mid parents(V)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}$)
      else return $\sum_v P(v \mid parents(V)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_v$)
         where $\mathbf{e}_v$ is $\mathbf{e}$ extended with $V = v$
```

Figure 13.11 The enumeration algorithm for exact inference in Bayes nets.

13.3.2 The variable elimination algorithm

The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in Figure 13.10. 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 (13.5) in *right-to-left* order (that is, *bottom up* in Figure 13.10). 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

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

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 $P(j\,|\,a)$ and $P(m\,|\,a)$ depend just on A because J and M are fixed by the query. They are therefore two-element vectors:

Factor

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

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

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

Pointwise product

Here the "×" operator is not ordinary matrix multiplication but instead the **pointwise product** operation, to be described shortly.

The evaluation process sums out variables (right to left) from pointwise products of factors to produce new factors, eventually yielding a factor that constitutes the solution—that is, 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

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

- 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

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

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

X	Y	$\mathbf{f}(X,Y)$	Y	Z	$\mathbf{g}(Y,Z)$	X	Y	Z	$\mathbf{h}(X,Y,Z)$
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 13.12 Illustrating pointwise multiplication: $\mathbf{f}(X,Y)\times\mathbf{g}(Y,Z)=\mathbf{h}(X,Y,Z)$.

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 **f** and **g** yields a new factor **h** whose variables are the *union* of the variables in **f** and **g** 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

$$\mathbf{f}(X_1\ldots X_j,Y_1\ldots Y_k)\times\mathbf{g}(Y_1\ldots Y_k,Z_1,\ldots Z_\ell)=\mathbf{h}(X_1\ldots X_j,Y_1\ldots Y_k,Z_1\ldots Z_\ell)$$

If all the variables are binary, then **f** and **g** have 2^{j+k} and $2^{k+\ell}$ entries, respectively, and the pointwise product has $2^{j+k+\ell}$ entries. For example, given two factors $\mathbf{f}(X,Y)$ and $\mathbf{g}(Y,Z)$, the pointwise product $\mathbf{f}\times\mathbf{g}=\mathbf{h}(X,Y,Z)$ has $2^{1+1+1}=8$ entries, as illustrated in Figure 13.12. 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.

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 X from $\mathbf{h}(X,Y,Z)$, we write

$$\begin{aligned}\mathbf{h}_2(Y,Z) &= \sum_x \mathbf{h}(X,Y,Z)=\mathbf{h}(x,Y,Z)+\mathbf{h}(\neg x,Y,Z)\\ &= \begin{pmatrix}.06 & .24\\ .42 & .28\end{pmatrix}+\begin{pmatrix}.18 & .72\\ .06 & .04\end{pmatrix}=\begin{pmatrix}.24 & .96\\ .48 & .32\end{pmatrix}.\end{aligned}$$

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, to sum out X from the product of **f** and **g**, we can move g outside the summation:

$$\sum_x \mathbf{f}(X,Y)\times\mathbf{g}(Y,Z)=\mathbf{g}(Y,Z)\times\sum_x \mathbf{f}(X,Y).$$

```
function ELIMINATION-ASK($X$, $\mathbf{e}$, $bn$) returns a distribution over $X$
  inputs: $X$, the query variable
          $\mathbf{e}$, observed values for variables $\mathbf{E}$
          $bn$, a Bayesian network with variables $vars$

  $factors \leftarrow [\,]$
  for each $V$ in ORDER($vars$) do
      $factors \leftarrow$ [MAKE-FACTOR($V$, $\mathbf{e}$)] + $factors$
      if $V$ is a hidden variable then $factors \leftarrow$ SUM-OUT($V$, $factors$)
  return NORMALIZE(POINTWISE-PRODUCT($factors$))
```

Figure 13.13 The variable elimination algorithm for exact inference in Bayes nets.

This is potentially much more efficient than computing the larger pointwise product **h** first and then summing X out from that.

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 13.13.

Variable ordering and variable relevance

The algorithm in Figure 13.13 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 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 (see Equation (13.5)), 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.

When applied to the insurance network shown in Figure 13.9, variable elimination shows considerable improvement over the naive enumeration algorithm. Using reverse topological order for the variables, exact inference using elimination is about 1,000 times faster than the enumeration algorithm.

13.3.3 The complexity of exact inference

The complexity of exact inference in Bayes nets depends strongly on the structure of the network. The burglary network of Figure 13.2 belongs to the family of networks in which there is at most one undirected path (i.e., ignoring the direction of the arrows) 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. These results hold for any ordering consistent with the topological ordering of the network (Exercise 13.VEEX).

Singly connected

Polytree

For **multiply connected** networks, such as the insurance network in Figure 13.9, 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 Bayes nets is NP-hard.* To prove this, we need to work out how to encode a propositional satisfiability problem as a Bayes net, such that running inference on this net tells us whether or not the original propositional sentences are satisfiable. (In the language of complexity theory, we **reduce** satisfiability problems to Bayes net inference problems.) This turns out to be quite straightforward. Figure 13.14 shows how to encode a particular 3-SAT problem. The propositional variables become the root variables of the network, each with prior probability 0.5. The next layer of nodes corresponds to the clauses, with each clause variable C_j connected to the appropriate variables as parents. The conditional distribution for a clause variable is a deterministic disjunction, with negation as needed, so that each clause variable is true if and only if the assignment to its parents satisfies that clause. Finally, S is the conjunction of the clause variables.

Multiply connected

Reduction

To determine if the original sentence is satisfiable, we simply evaluate $P(S = true)$. If the sentence is *satisfiable*, then there is some possible assignment to the logical variables that makes S true; in the Bayes net, this means that there is a possible world with nonzero probability in which the root variables have that assignment, the clause variables have value *true*, and S has value *true*. Therefore, $P(S = true) > 0$ for a satisfiable sentence. Conversely, $P(S = true) = 0$ for an unsatisfiable sentence: all worlds with $S = true$ have probability 0. Hence, we can use Bayes net inference to solve 3-SAT problems; from this, we conclude that Bayes net inference is NP-hard.

We can, in fact, do more than this. Notice that the probability of each satisfying assignment is 2^{-n} for a problem with n variables. Hence, the *number* of satisfying assignments is $P(S = true)/(2^{-n})$. Because computing the *number* of satisfying assignments for a 3-SAT

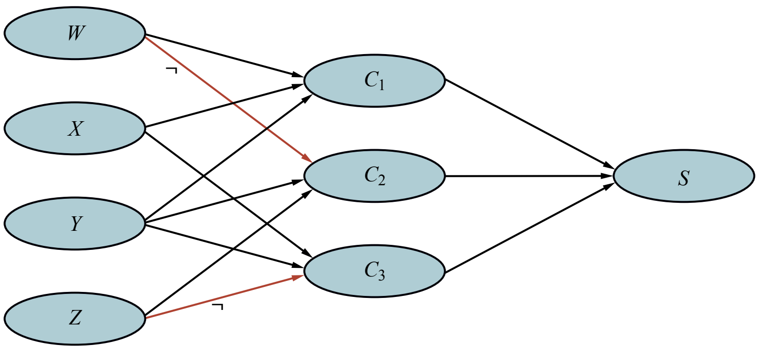


Figure 13.14 Bayes net encoding of the 3-CNF sentence

$$(W \lor X \lor Y) \land (\neg W \lor Y \lor Z) \land (X \lor Y \lor \neg Z)\,.$$

problem is #P-complete ("number-P complete"), this means that Bayes net inference is #P-hard—that is, strictly harder than NP-complete problems.

There is a close connection between the complexity of Bayes net 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 "treelike" its constraint graph is. Measures such as **tree width**, which bound the complexity of solving a CSP, can also be applied directly to Bayes nets. Moreover, the variable elimination algorithm can be generalized to solve CSPs as well as Bayes nets.

As well as reducing satisfiability problems to Bayes net inference, we can reduce Bayes net inference to satisfiability, which allows us to take advantage of the powerful machinery developed for SAT-solving (see Chapter 7). In this case, the reduction is to a particular form of SAT solving called **weighted model counting** (WMC). Regular model counting counts the number of satisfying assignments for a SAT expression; WMC sums the total weight of those satisfying assignments—where, in this application, the weight is essentially the product of the conditional probabilities for each variable assignment given its parents. (See Exercise 13.WMCX for details.) Partly because SAT-solving technology has been so well optimized for large-scale applications, Bayes net inference via WMC is competitive with and sometimes superior to other exact algorithms on networks with large tree width.

Weighted model counting

13.3.4 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 Bayes net 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 13.15(a) can be converted into a polytree by combining the *Sprinkler* and *Rain* node into a cluster node called *Sprinkler+Rain*, as shown in Fig-

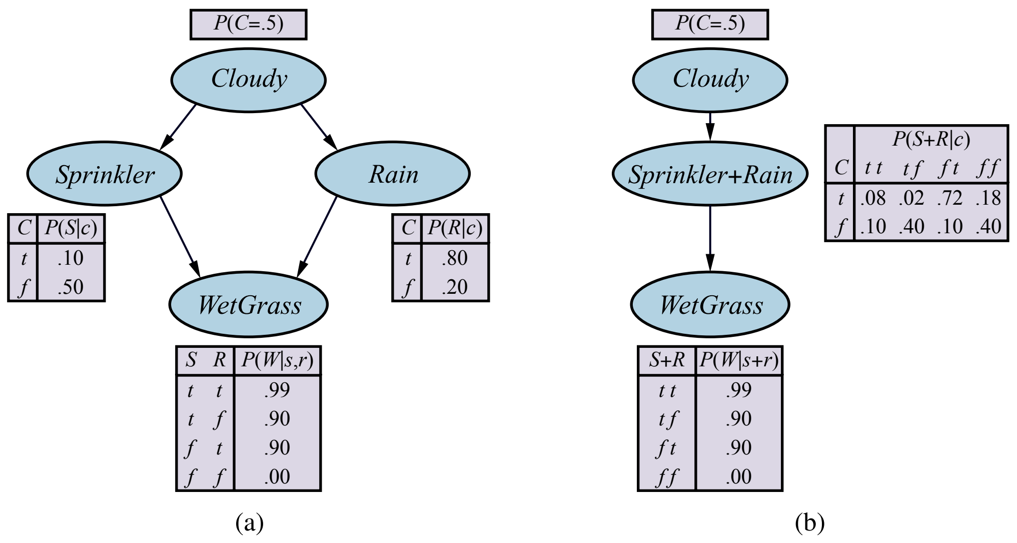


Figure 13.15 (a) A multiply connected network describing Mary's daily lawn routine: each morning, she checks the weather; if it's cloudy, she usually doesn't turn on the sprinkler; if the sprinkler is on, or if it rains during the day, the grass will be wet. Thus, *Cloudy* affects *WetGrass* via two different causal pathways. (b) A clustered equivalent of the multiply connected network.

ure 13.15(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.

Meganode

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 5) 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.

13.4 Approximate Inference for Bayesian Networks

Given the intractability of exact inference in large networks, we will now 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. They work by generating random events based on the probabilities in the Bayes net and counting up the different answers found in those random events. With enough samples, we can get arbitrarily close to recovering the true probability distribution—provided the Bayes net has no deterministic conditional distributions.

Monte Carlo

Monte Carlo algorithms, of which simulated annealing (page 133) 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 in Bayes nets. We describe two families of algorithms: direct sampling and Markov chain sampling. Several other approaches for approximate inference are mentioned in the notes at the end of the chapter.

13.4.1 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 r uniformly distributed in the range $[0,1]$, it is a simple matter to sample any distribution on a single variable, whether discrete or continuous. This is done by constructing the cumulative distribution for the variable and returning the first value whose cumulative probability exceeds r (see Exercise 13.PRSA).

We begin with a random sampling process for a Bayes net 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. (Because we sample in topological order, the parents are guaranteed to have values already.) This algorithm is shown in Figure 13.16. Applying it to the network in Figure 13.15(a) with the ordering *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*, we might produce a random event as follows:

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 $[true, false, true, true]$.

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

```
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 each variable $X_i$ in $X_1, \ldots, X_n$ do
      $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$
```

Figure 13.16 A sampling algorithm that generates events from a Bayesian network. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents.

generated by the PRIOR-SAMPLE algorithm. Just looking at the sampling process, we have

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

because each sampling step depends only on the parent values. This expression should look familiar, because it is also the probability of the event according to the Bayesian net's representation of the joint distribution, as stated in Equation (13.2). That is, we have

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

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

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

$$\lim_{N\to\infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n)\,. \tag{13.6}$$

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 $m \le n$, as follows:

Consistent

$$P(x_1, \ldots, x_m) \approx N_{PS}(x_1, \ldots, x_m)/N\,. \tag{13.7}$$

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. We will use $\hat{P}$ (pronounced "P-hat") to mean an estimated probability. So, if we generate 1,000 samples from the sprinkler network, and 511 of them have $Rain = true$, then the estimated probability of rain is $\hat{P}(Rain = true) = 0.511$.

Rejection sampling in Bayesian networks

Rejection sampling is a general method for producing samples from a hard-to-sample distribution given an easy-to-sample distribution. In its simplest form, it can be used to compute conditional probabilities—that is, to determine $P(X \mid \mathbf{e})$. The REJECTION-SAMPLING algorithm is shown in Figure 13.17. First, it generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate $\hat{P}(X = x \mid \mathbf{e})$ is obtained by counting how often $X = x$ occurs in the remaining samples.

Rejection sampling

Let $\hat{\mathbf{P}}(X \mid \mathbf{e})$ be the estimated distribution that the algorithm returns; this distribution is computed by normalizing $\mathbf{N}_{PS}(X, \mathbf{e})$, the vector of sample counts for each value of X where the sample agrees with the evidence $\mathbf{e}$:

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

```
function REJECTION-SAMPLING($X, \mathbf{e}, bn, N$) returns an estimate of $\mathbf{P}(X \mid \mathbf{e})$
  inputs: $X$, the query variable
          $\mathbf{e}$, observed values for variables $\mathbf{E}$
          $bn$, a Bayesian network
          $N$, the total number of samples to be generated
  local variables: $\mathbf{C}$, a vector of counts for each value of $X$, initially zero

  for $j$ = 1 to $N$ do
     $\mathbf{x} \leftarrow$ PRIOR-SAMPLE($bn$)
     if $\mathbf{x}$ is consistent with $\mathbf{e}$ then
        $\mathbf{C}[j] \leftarrow \mathbf{C}[j]+1$ where $x_j$ is the value of $X$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{C}$)
```

Figure 13.17 The rejection-sampling algorithm for answering queries given evidence in a Bayesian network.

From Equation (13.7), this becomes

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

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

Continuing with our example from Figure 13.15(a), let us assume that we wish to estimate $\mathbf{P}(Rain \mid Sprinkler = true)$, using 100 samples. Of the 100 that we generate, suppose that 73 have $Sprinkler = false$ and are rejected, while 27 have $Sprinkler = true$; of the 27, 8 have $Rain = true$ and 19 have $Rain = false$. Hence,

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

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

Now we know that rejection sampling converges to the correct answer, the next question is, how fast does that happen? More precisely, how many samples are required before we know that the resulting estimates are close to the correct answers with high probability? Whereas the complexity of exact algorithms depends to a large extent on the topology of the network—trees are easy, densely connected networks are hard—the complexity of rejection sampling depends primarily on the fraction of samples that are accepted. This fraction is exactly equal to the prior probability of the evidence, $P(\mathbf{e})$. Unfortunately, for complex problems with many evidence variables, this fraction is vanishingly small. When applied to the discrete version of the car insurance network in Figure 13.9, the fraction of samples consistent with a typical evidence case sampled from the network itself is usually between one in a thousand and one in ten thousand. Convergence is extremely slow (see Figure 13.19 below).

We expect the fraction of samples consistent with the evidence $\mathbf{e}$ to drop exponentially as the number of evidence variables grows, so the procedure is unusable for complex problems. It also has difficulties with continuous-valued evidence variables, because the probability of producing a sample consistent with such evidence is zero (if it is really continuous-valued) or infinitesimal (if it is merely a finite-precision floating-point number).

Notice that rejection sampling is very similar to the estimation of conditional probabilities in the real world. For example, to estimate the conditional probability that any humans survive after a 1km-diameter asteroid crashes into the Earth, one can simply count how often any humans survive after a 1km-diameter asteroid crashes into the Earth, ignoring all those days when no such event occurs. (Here, the universe itself plays the role of the sample-generation algorithm.) To get a decent estimate, one might need to wait for 100 such events to occur. Obviously, this could take a long time, and that is the weakness of rejection sampling.

Importance sampling

Importance sampling

The general statistical technique of **importance sampling** aims to emulate the effect of sampling from a distribution P using samples from another distribution Q. We ensure that the answers are correct in the limit by applying a correction factor $P(\mathbf{x})/Q(\mathbf{x})$, also known as a **weight**, to each sample $\mathbf{x}$ when counting up the samples.

The reason for using importance sampling in Bayes nets is simple: we would like to sample from the true posterior distribution conditioned on all the evidence, but usually this is too hard;[6] so instead, we sample from an easy distribution and apply the necessary corrections. The reason why importance sampling works is also simple. Let the nonevidence variables be $\mathbf{Z}$. If we could sample directly from $P(\mathbf{z}\,|\,\mathbf{e})$, we could construct estimates like this:

$$\hat{P}(\mathbf{z}\,|\,\mathbf{e}) = \frac{N_P(\mathbf{z})}{N} \approx P(\mathbf{z}\,|\,\mathbf{e})$$

where $N_P(\mathbf{z})$ is the number of samples with $\mathbf{Z}=\mathbf{z}$ when sampling from P. Now suppose instead that we sample from $Q(\mathbf{z})$. The estimate in this case includes the correction factors:

$$\hat{P}(\mathbf{z}\,|\,\mathbf{e}) = \frac{N_Q(\mathbf{z})}{N}\frac{P(\mathbf{z}\,|\,\mathbf{e})}{Q(\mathbf{z})} \approx Q(\mathbf{z})\frac{P(\mathbf{z}\,|\,\mathbf{e})}{Q(\mathbf{z})} = P(\mathbf{z}\,|\,\mathbf{e})\,.$$

Thus, the estimate converges to the correct value *regardless of which sampling distribution Q is used.* (The only technical requirement is that $Q(\mathbf{z})$ should not be zero for any $\mathbf{z}$ where $P(\mathbf{z}\,|\,\mathbf{e})$ is nonzero.) Intuitively, the correction factor compensates for oversampling or undersampling. For example, if $Q(\mathbf{z})$ is much bigger than $P(\mathbf{z}\,|\,\mathbf{e})$ for some $\mathbf{z}$, then there will be many more samples of that $\mathbf{z}$ than there should be, but each will have a small weight, so it works out just as if there were the right number.

Likelihood weighting

As for which Q to use, we want one that is easy to sample from and as close as possible to the true posterior $P(\mathbf{z}\,|\,\mathbf{e})$. The most common approach is called **likelihood weighting** (for reasons we will see shortly). As shown in the WEIGHTED-SAMPLE function in Figure 13.18, the algorithm fixes the values for the evidence variables $\mathbf{E}$ and samples all the nonevidence variables in topological order, each conditioned on its parents. This guarantees that each event generated is consistent with the evidence.

Let's call the sampling distribution produced by this algorithm Q_{WS}. If the nonevidence variables are $\mathbf{Z}=\{Z_1,\ldots,Z_l\}$, then we have

$$Q_{WS}(\mathbf{z}) = \prod_{i=1}^{l} P(z_i\,|\,parents(Z_i)) \tag{13.8}$$

[6] If it was easy, 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 $\mathbf{P}(X \mid \mathbf{e})$
  inputs: $X$, the query variable
          $\mathbf{e}$, observed values for variables $\mathbf{E}$
          $bn$, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
          $N$, the total number of samples to be generated
  local variables: $\mathbf{W}$, a vector of weighted counts for each value of $X$, initially zero

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

function WEIGHTED-SAMPLE($bn, \mathbf{e}$) returns an event and a weight
  $w \leftarrow 1$; $\mathbf{x} \leftarrow$ an event with $n$ elements, with values fixed from $\mathbf{e}$
  for $i$ = 1 to $n$ do
     if $X_i$ is an evidence variable with value $x_{ij}$ in $\mathbf{e}$
        then $w \leftarrow w \times P(X_i = x_{ij} \mid parents(X_i))$
        else $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$, $w$
```

Figure 13.18 The likelihood-weighting algorithm for inference in Bayesian networks. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

because each variable is sampled conditioned on its parents. In order to complete the algorithm, we need to know how to compute the weight for each sample generated from Q_{WS}. According to the general scheme for importance sampling, the weight should be

$$w(\mathbf{z}) = P(\mathbf{z} \mid \mathbf{e})/Q_{WS}(\mathbf{z}) = \alpha P(\mathbf{z}, \mathbf{e})/Q_{WS}(\mathbf{z})$$

where the normalizing factor $\alpha = 1/P(\mathbf{e})$ is the same for all samples. Now $\mathbf{z}$ and $\mathbf{e}$ together cover all the variables in the Bayes net, so $P(\mathbf{z}, \mathbf{e})$ is just the product of all the conditional probabilities (Equation (13.2) page 433); and we can write this as the product of the conditional probabilities for the nonevidence variables times the product of the conditional probabilities for the evidence variables:

$$\begin{aligned} w(\mathbf{z}) &= \alpha \frac{P(\mathbf{z}, \mathbf{e})}{Q_{WS}(\mathbf{z})} = \alpha \frac{\prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \prod_{i=1}^{m} P(e_i \mid parents(E_i))}{\prod_{i=1}^{l} P(z_i \mid parents(Z_i))} \\ &= \alpha \prod_{i=1}^{m} P(e_i \mid parents(E_i)) \,. \end{aligned} \tag{13.9}$$

Thus the weight is the product of the conditional probabilities for the evidence variables given their parents. (Probabilities of evidence are generally called **likelihoods**, hence the name.) The weight calculation is implemented incrementally in WEIGHTED-SAMPLE, multiplying by the conditional probability each time an evidence variable is encountered. The normalization is done at the end before the query result is returned.

Let us apply the algorithm to the network shown in Figure 13.15(a), with the query $\mathbf{P}(Rain \mid Cloudy = true, WetGrass = true)$ and the ordering *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*.

(Any topological ordering will do.) The process goes as follows: First, the weight w is set to 1.0. Then an event is generated:

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

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

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

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 "downstream"—that is, late in the variable ordering—because then the nonevidence variables will have no evidence in their parents and ancestors to guide the generation of samples. This means the samples will be mere hallucinations—simulations that bear little resemblance to the reality suggested by the evidence.

When applied to the discrete version of the car insurance network in Figure 13.9, likelihood weighting is considerably more efficient than rejection sampling (see Figure 13.19). The insurance network is a relatively benign case for likelihood weighting because much of the evidence is "upstream" and the query variables are leaf nodes of the network.

13.4.2 Inference by Markov chain simulation

Markov chain Monte Carlo

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

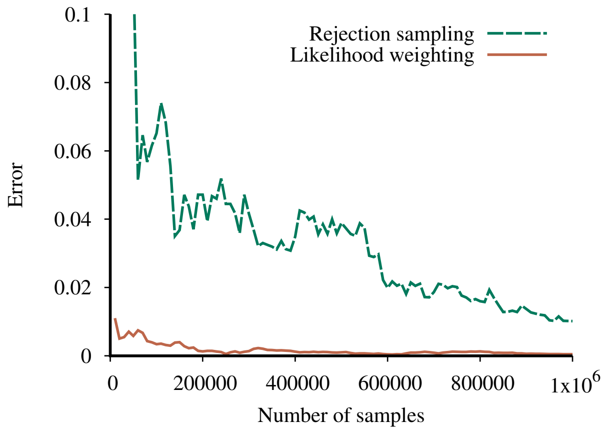


Figure 13.19 Performance of rejection sampling and likelihood weighting on the insurance network. The x-axis shows the number of samples generated and the y-axis shows the maximum absolute error in any of the probability values for a query on *PropertyCost*.

Markov chain

Gibbs sampling

Metropolis–Hastings

The term **Markov chain** refers to a random process that generates a sequence of states. (Markov chains also figure prominently in Chapters 14 and 16; the simulated annealing algorithm in Chapter 4 and the WALKSAT algorithm in Chapter 7 are also members of the MCMC family.) We begin by describing a particular form of MCMC called **Gibbs sampling**, which is especially well suited for Bayes nets. We then describe the more general **Metropolis–Hastings** algorithm, which allows much greater flexibility in generating samples.

Gibbs sampling in Bayesian networks

The Gibbs sampling algorithm for Bayesian networks starts with an arbitrary state (with the evidence variables fixed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables X_i. Recall from page 437 that X_i is independent of all other variables given its Markov blanket (its parents, children, and children's other parents); therefore, Gibbs sampling for X_i means sampling *conditioned on the current values of the variables in its Markov blanket*. The algorithm wanders randomly around the state space—the space of possible complete assignments—flipping one variable at a time, but keeping the evidence variables fixed. The complete algorithm is shown in Figure 13.20.

Consider the query $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true)$ for the network in Figure 13.15(a). The evidence variables *Sprinkler* and *WetGrass* are fixed to their observed values (both *true*), and the nonevidence variables *Cloudy* and *Rain* are initialized randomly to, say, *true* and *false* respectively. Thus, the initial state is $[true, \mathbf{true}, false, \mathbf{true}]$, where we have marked the fixed evidence values in bold. Now the nonevidence variables Z_i are sampled repeatedly in some random order according to a probability distribution $\rho(i)$ for choosing variables. For example:

1. *Cloudy* is chosen and then sampled, given the current values of its Markov blanket: in this case, we sample from $\mathbf{P}(Cloudy \mid Sprinkler = true, Rain = false)$. Suppose the result is $Cloudy = false$. Then the new current state is $[false, \mathbf{true}, false, \mathbf{true}]$.

```
function GIBBS-ASK($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $\mathbf{P}(X \mid \mathbf{e})$
  local variables: $\mathbf{C}$, a vector of counts for each value of $X$, initially zero
                   $\mathbf{Z}$, the nonevidence variables in $bn$
                   $\mathbf{x}$, the current state of the network, initialized from $\mathbf{e}$

  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Z}$
  for $k = 1$ to $N$ do
      choose any variable $Z_i$ from $\mathbf{Z}$ according to any distribution $\rho(i)$
      set the value of $Z_i$ in $\mathbf{x}$ by sampling from $\mathbf{P}(Z_i \mid mb(Z_i))$
      $\mathbf{C}[j] \leftarrow \mathbf{C}[j] + 1$ where $x_j$ is the value of $X$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{C}$)
```

Figure 13.20 The Gibbs sampling algorithm for approximate inference in Bayes nets; this version chooses variables at random, but cycling through the variables but also works.

2. *Rain* is chosen and then sampled, given the current values of its Markov blanket: in this case, we sample from $\mathbf{P}(Rain \mid Cloudy = false, Sprinkler = true, WetGrass = true)$. Suppose this yields $Rain = true$. The new current state is $[false, \mathbf{true}, true, \mathbf{true}]$.

The one remaining detail concerns the method of calculating the Markov blanket distribution $\mathbf{P}(X_i \mid mb(X_i))$, where $mb(X_i)$ denotes the values of the variables in X_i's Markov blanket, $MB(X_i)$. Fortunately, this does not involve any complex inference. As shown in Exercise 13.MARB, the distribution is given by

$$P(x_i \mid mb(X_i)) = \alpha P(x_i \mid parents(X_i)) \prod_{Y_j \in Children(X_i)} P(y_j \mid parents(Y_j)) . \tag{13.10}$$

In other words, for each value x_i, the probability is given by multiplying probabilities from the CPTs of X_i and its children. For example, in the first sampling step shown above, we sampled from $\mathbf{P}(Cloudy \mid Sprinkler = true, Rain = false)$. By Equation (13.10), and abbreviating the variable names, we have

$$\begin{aligned} P(c \mid s, \neg r) &= \alpha P(c) P(s \mid c) P(\neg r \mid c) = \alpha\, 0.5 \cdot 0.1 \cdot 0.2 \\ P(\neg c \mid s, \neg r) &= \alpha P(\neg c) P(s \mid \neg c) P(\neg r \mid \neg c) = \alpha\, 0.5 \cdot 0.5 \cdot 0.8 , \end{aligned}$$

so the sampling distribution is $\alpha \langle 0.001, 0.020 \rangle \approx \langle 0.048, 0.952 \rangle$.

Figure 13.21(a) shows the complete Markov chain for the case where variables are chosen uniformly, i.e., $\rho(Cloudy) = \rho(Rain) = 0.5$. The algorithm is simply wandering around in this graph, following links with the stated probabilities. Each state visited during this process is a sample that contributes to the estimate for the query variable *Rain*. If the process visits 20 states where *Rain* is true and 60 states where *Rain* is false, then the answer to the query is NORMALIZE$(\langle 20, 60 \rangle) = \langle 0.25, 0.75 \rangle$.

Analysis of Markov chains

We have said that Gibbs sampling works by wandering randomly around the state space to generate samples. To explain why Gibbs sampling works *correctly*—that is, why its estimates converge to correct values in the limit—we will need some careful analysis. (This section is somewhat mathematical and can be skipped on first reading.)

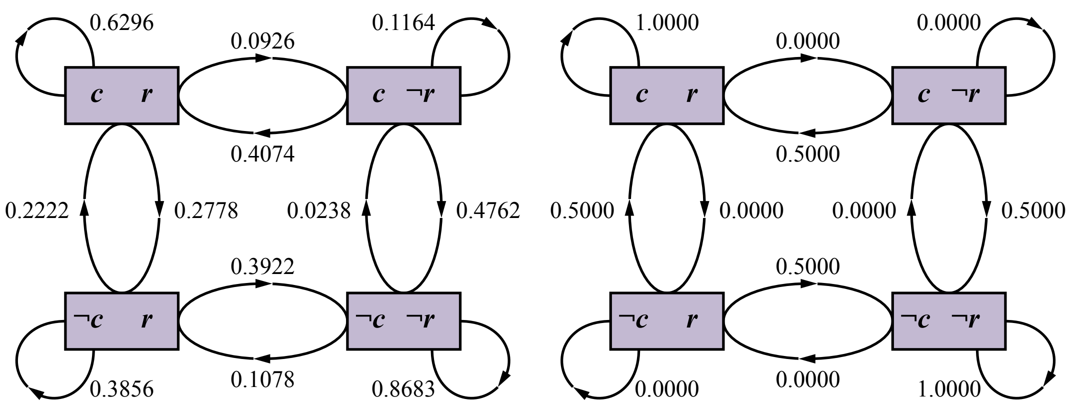


Figure 13.21 (a) The states and transition probabilities of the Markov chain for the query $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true)$. Note the self-loops: the state stays the same when *either* variable is chosen and then resamples the same value it already has. (b) The transition probabilities when the CPT for *Rain* constrains it to have the same value as *Cloudy*.

Transition kernel

We begin with some of the basic concepts for analyzing Markov chains in general. Any such chain is defined by its initial state and its **transition kernel** $k(\mathbf{x} \to \mathbf{x}')$—the probability of a transition to state $\mathbf{x}'$ starting from state $\mathbf{x}$. Now suppose that we run the Markov chain for t steps, and let $\pi_t(\mathbf{x})$ be the probability that the system is in state $\mathbf{x}$ at time t. Similarly, let $\pi_{t+1}(\mathbf{x}')$ be the probability of being in state $\mathbf{x}'$ at time $t+1$. Given $\pi_t(\mathbf{x})$, we can calculate $\pi_{t+1}(\mathbf{x}')$ by summing, for all states $\mathbf{x}$ the system could be in at time t, the probability of being in $\mathbf{x}$ times the probability of making the transition to $\mathbf{x}'$:

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

Stationary distribution

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

$$\pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) k(\mathbf{x} \to \mathbf{x}') \qquad \text{for all } \mathbf{x}' . \tag{13.11}$$

Ergodic

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

Equation (13.11) can be read as saying that the expected "outflow" from each state (i.e., its current "population") is equal to the expected "inflow" from all the states. One obvious way to satisfy this relationship is if the expected flow between any pair of states is the same in both directions; that is,

$$\pi(\mathbf{x}) k(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}') k(\mathbf{x}' \to \mathbf{x}) \qquad \text{for all } \mathbf{x},\ \mathbf{x}' . \tag{13.12}$$

Detailed balance

When these equations hold, we say that $k(\mathbf{x} \to \mathbf{x}')$ is in **detailed balance** with $\pi(\mathbf{x})$. One special case is the self-loop $\mathbf{x} = \mathbf{x}'$, i.e., a transition from a state to itself. In that case, the detailed balance condition becomes $\pi(\mathbf{x}) k(\mathbf{x} \to \mathbf{x}) = \pi(\mathbf{x}) k(\mathbf{x} \to \mathbf{x})$ which is of course trivially true for any stationary distribution π and any transition kernel k.

We can show that detailed balance implies stationarity simply by summing over $\mathbf{x}$ in Equation (13.12). We have

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

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

Why Gibbs sampling works

We will now show that Gibbs sampling returns consistent estimates for posterior probabilities. The basic claim is straightforward: *the stationary distribution of the Gibbs sampling process is exactly the posterior distribution for the nonevidence variables conditioned on the evidence.* This remarkable property follows from the specific way in which the Gibbs sampling process moves from state to state.

The general definition of Gibbs sampling is that a variable X_i is chosen and then sampled conditionally on the current values of *all* the other variables. (When applied specifically to Bayes nets, we simply use the additional fact that sampling conditionally on all variables is equivalent to sampling conditionally on the variable's Markov blanket, as shown on page 437.) We will use the notation $\overline{\mathbf{X}_i}$ to refer to these other variables (except the evidence variables); their values in the current state are $\overline{\mathbf{x}_i}$.

To write down the transition kernel $k(\mathbf{x} \to \mathbf{x}')$ for Gibbs sampling, there are three cases to consider:

1. The states $\mathbf{x}$ and $\mathbf{x}'$ differ in two or more variables. In that case, $k(\mathbf{x} \to \mathbf{x}') = 0$ because Gibbs sampling changes only a single variable.
2. The states differ in exactly one variable X_i that changes its value from x_i to x_i'. The probability of such an occurrence is
$$k(\mathbf{x} \to \mathbf{x}') = k((x_i, \overline{\mathbf{x}_i}) \to (x_i', \overline{\mathbf{x}_i})) = \rho(i)P(x_i' \,|\, \overline{\mathbf{x}_i}) \,. \tag{13.13}$$
3. The states are the same: $\mathbf{x} = \mathbf{x}'$. In that case, *any* variable could be chosen but then the sampling process produces the same value the variable already has. The probability of such an occurrence is
$$k(\mathbf{x} \to \mathbf{x}) = \sum_i \rho(i)k((x_i, \overline{\mathbf{x}_i}) \to (x_i, \overline{\mathbf{x}_i})) = \sum_i \rho(i)P(x_i \,|\, \overline{\mathbf{x}_i}) \,.$$

Now we show that this general definition of Gibbs sampling satisfies the detailed balance equation with a stationary distribution equal to $P(\mathbf{x} \,|\, \mathbf{e})$, the true posterior distribution on the nonevidence variables. That is, we show that $\pi(\mathbf{x})k(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}')k(\mathbf{x}' \to \mathbf{x})$ where $\pi(\mathbf{x}) = P(\mathbf{x} \,|\, \mathbf{e})$, for all states $\mathbf{x}$ and $\mathbf{x}'$.

For the first and third cases given above, detailed balance is *always* satisfied: if two states differ in two or more variables, the transition probability in both directions is zero. If $\mathbf{x} \neq \mathbf{x}'$ then from Equation (13.13), we have

$$\begin{aligned}
\pi(\mathbf{x})k(\mathbf{x} \to \mathbf{x}') &= P(\mathbf{x} \,|\, \mathbf{e})\rho(i)P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) = \rho(i)P(x_i, \overline{\mathbf{x}_i} \,|\, \mathbf{e})P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) \\
&= \rho(i)P(x_i \,|\, \overline{\mathbf{x}_i}, \mathbf{e})P(\overline{\mathbf{x}_i} \,|\, \mathbf{e})P(x_i' \,|\, \overline{\mathbf{x}_i}, \mathbf{e}) \qquad \text{(using the chain rule on the first term)} \\
&= \rho(i)P(x_i \,|\, \overline{\mathbf{x}_i}, \mathbf{e})P(x_i', \overline{\mathbf{x}_i} \,|\, \mathbf{e}) \qquad \text{(reverse chain rule on last two terms)} \\
&= \pi(\mathbf{x}')k(\mathbf{x}' \to \mathbf{x}) \,.
\end{aligned}$$

The final piece of the puzzle is the ergodicity of the chain—that is, every state must be reachable from every other and there are no periodic cycles. Both conditions are satisfied provided

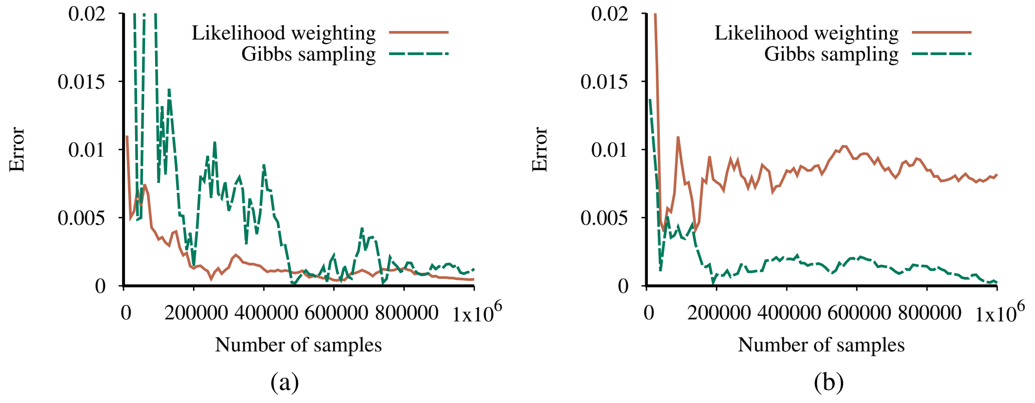


Figure 13.22 Performance of Gibbs sampling compared to likelihood weighting on the car insurance network: (a) for the standard query on *PropertyCost*, and (b) for the case where the output variables are observed and *Age* is the query variable.

the CPTs do not contain probabilities of 0 or 1. Reachability comes from the fact that we can convert one state into another by changing one variable at a time, and the absence of periodic cycles comes from the fact that every state has a self-loop with nonzero probability. Hence, under the stated conditions, k is ergodic, which means that the samples generated by Gibbs sampling will eventually be drawn from the true posterior distribution.

Complexity of Gibbs sampling

First, the good news: each Gibbs sampling step involves calculating the Markov blanket distribution for the chosen variable X_i, which requires a number of multiplications proportional to the number of X_i's children and the size of X_i's range. This is important because it means that *the work required to generate each sample is independent of the size of the network.*

Now, the not necessarily bad news: the complexity of Gibbs sampling is much harder to analyze than that of rejection sampling and likelihood weighting. The first thing to notice is that Gibbs sampling, unlike likelihood weighting, *does* pay attention to downstream evidence. Information propagates from evidence nodes in all directions: first, any neighbors of the evidence nodes sample values that reflect the evidence in those nodes; then *their* neighbors, and so on. Thus, we expect Gibbs sampling to outperform likelihood weighting when evidence is mostly downstream; and indeed, this is borne out in Figure 13.22.

Mixing rate

The rate of convergence for Gibbs sampling—the **mixing rate** of the Markov chain defined by the algorithm—depends strongly on the quantitative properties of the conditional distributions in the network. To see this, consider what happens in Figure 13.15(a) as the CPT for *Rain* becomes deterministic: it rains *if and only if* it is cloudy. In that case, the true posterior distribution for the query $\mathbf{P}(Rain \mid sprinkler, wetGrass)$ is roughly $\langle 0.18, 0.82 \rangle$ but Gibbs sampling will never reach this value. The problem is that the only two joint states for *Cloudy* and *Rain* that have non-zero probability are $[true, true]$ and $[false, false]$. Starting in $[true, true]$, the chain can never reach $[false, false]$ because transitions to the intermediate states have probability zero (see Figure 13.21(b)). So, if the process starts in $[true, true]$ it always reports a posterior probability for the query of $\langle 1.0, 0.0\rangle$; if it starts in [*false*,*false*] it always reports a posterior probability for the query of $\langle 0.0, 1.0\rangle$.

Gibbs sampling fails in this case because the deterministic relationship between *Cloudy* and *Rain* breaks the property of ergodicity that is required for convergence. If, however, we make the relationship *nearly* deterministic, then convergence is restored, but happens arbitrarily slowly. There are several fixes that help MCMC algorithms mix more quickly. One is **block sampling**: sampling multiple variables simultaneously. In this case, we could sample *Cloudy* and *Rain* jointly, conditioned on their combined Markov blanket. Another is to generate next states more intelligently, as we will see in the next section.

Block sampling

Metropolis–Hastings sampling

The Metropolis–Hastings or MH sampling method is perhaps the most broadly applicable MCMC algorithm. Like Gibbs sampling, MH is designed to generate samples $\mathbf{x}$ (eventually) according to target probabilities $\pi(\mathbf{x})$; in the case of inference in Bayesian networks, we want $\pi(\mathbf{x}) = P(\mathbf{x} \mid \mathbf{e})$. Like simulated annealing (page 133), MH has two stages in each iteration of the sampling process:

1. Sample a new state $\mathbf{x}'$ from a **proposal distribution** $q(\mathbf{x}' \mid \mathbf{x})$, given the current state $\mathbf{x}$.
2. Probabilistically accept or reject $\mathbf{x}'$ according to the **acceptance probability**

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

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

Proposal distribution

Acceptance probability

The transition kernel for MH consists of this two-step process. Note that if the proposal is rejected, the chain stays in the same state.

The proposal distribution is responsible, as its name suggests, for proposing a next state $\mathbf{x}'$. For example, $q(\mathbf{x}' \mid \mathbf{x})$ could be defined as follows:

- With probability 0.95, perform a Gibbs sampling step to generate $\mathbf{x}'$.
- Otherwise, generate $\mathbf{x}'$ by running the WEIGHTED-SAMPLE algorithm from page 458.

This proposal distribution causes MH to do about 20 steps of Gibbs sampling then "restarts" the process from a new state (assuming it is accepted) that is generated from scratch. By this stratagem, it gets around the problem of Gibbs sampling getting stuck in one part of the state space and being unable to reach the other parts.

You might ask how on Earth we know that MH with such a weird proposal actually converges to the right answer. The remarkable thing about MH is that *convergence to the correct stationary distribution is guaranteed for* any *proposal distribution,* provided the resulting transition kernel is ergodic.

This property follows from the way the acceptance probability is defined. As with Gibbs sampling, the self-loop with $\mathbf{x} = \mathbf{x}'$ automatically satisfies detailed balance, so we focus on the case where $\mathbf{x} \neq \mathbf{x}'$. This can occur only if the proposal is accepted. The probability of such a transition occurring is

$$k(\mathbf{x} \to \mathbf{x}') = q(\mathbf{x}' \mid \mathbf{x})a(\mathbf{x}' \mid \mathbf{x}).$$

As with Gibbs sampling, proving detailed balance means showing that the flow from $\mathbf{x}$ to $\mathbf{x}'$, $\pi(\mathbf{x})k(\mathbf{x} \to \mathbf{x}')$, matches the flow from $\mathbf{x}'$ to $\mathbf{x}$, $\pi(\mathbf{x}')k(\mathbf{x}' \to \mathbf{x})$. After plugging in the expression above for $k(\mathbf{x} \to \mathbf{x}')$, the proof is quite straightforward:

$$\begin{aligned}
\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x})a(\mathbf{x}' \mid \mathbf{x}) &= \pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x}) \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x})}\right) && \text{(definition of } a(\cdot \mid \cdot)\text{)} \\
&= \min\left(\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x}), \pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')\right) && \text{(multiplying in)} \\
&= \pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}') \min\left(\frac{\pi(\mathbf{x})q(\mathbf{x}' \mid \mathbf{x})}{\pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')}, 1\right) && \text{(dividing out)} \\
&= \pi(\mathbf{x}')q(\mathbf{x} \mid \mathbf{x}')a(\mathbf{x} \mid \mathbf{x}') .
\end{aligned}$$

Mathematical properties aside, the important part of MH to focus on is the ratio $\pi(\mathbf{x}')/\pi(\mathbf{x})$ in the acceptance probability. This says that if a next state is proposed that is *more* likely than the current state, it will definitely be accepted. (We are overlooking, for now, the term $q(\mathbf{x} \mid \mathbf{x}')/q(\mathbf{x}' \mid \mathbf{x})$, which is there to ensure detailed balance and is, in many state spaces, equal to 1 because of symmetry.) If the proposed state is *less* likely than the current state, its probability of being accepted drops proportionally.

Thus, one guideline for designing proposal distributions is to make sure the new states being proposed are reasonably likely. Gibbs sampling does this automatically: it proposes from the Gibbs distribution $P(X_i \mid \overline{\mathbf{x}_i})$, which means that the probability of generating any particular new value for X_i is directly proportional to its probability. (Exercise 13.GIBM asks you to show that Gibbs is a special case of MH with an acceptance probability of 1.)

Another guideline is to make sure that the chain mixes well, which means sometimes proposing large moves to distant parts of the state space. In the example given above, the occasional use of WEIGHTED-SAMPLE to restart the chain in a new state serves this purpose.

Besides near-complete freedom in designing proposal distributions, MH has two additional properties that make it practical. First, the posterior probability $\pi(\mathbf{x}) = P(\mathbf{x} \mid \mathbf{e})$ appears in the acceptance calculation only in the form of a ratio $\pi(\mathbf{x}')/\pi(\mathbf{x})$, which is very fortunate. Computing $P(\mathbf{x} \mid \mathbf{e})$ directly is the very computation we're trying to approximate using MH, so it wouldn't make sense to do it for each sample! Instead, we use the following trick:

$$\frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} = \frac{P(\mathbf{x}' \mid \mathbf{e})}{P(\mathbf{x} \mid \mathbf{e})} = \frac{P(\mathbf{x}', \mathbf{e})}{P(\mathbf{e})} \frac{P(\mathbf{e})}{P(\mathbf{x}, \mathbf{e})} = \frac{P(\mathbf{x}', \mathbf{e})}{P(\mathbf{x}, \mathbf{e})} .$$

The terms in this ratio are full joint probabilities, i.e., products of conditional probabilities in the Bayes net. The second useful property of this ratio is that as long as the proposal distribution makes only local changes in $\mathbf{x}$ to produce $\mathbf{x}'$, only a small number of terms in the product of conditional probabilities will be different. All of the conditional probabilities involving variables whose values are unchanged will cancel out in the ratio. So, as with Gibbs sampling, the work required to generate each sample is independent of the size of the network as long as the state changes are local.

13.4.3 Compiling approximate inference

The sampling algorithms in Figures 13.17, 13.18, and 13.20 share a common property: they operate on a Bayes net represented as a data structure. This seems quite natural: after all, a Bayes net is a directed acyclic graph, so how else could it be represented? The problem with this approach is that the operations required to access the data structure—for example to find a node's parents—are repeated thousands or millions of times as the sampling algorithm runs, and *all of these computations are completely unnecessary*.

The network's structure and conditional probabilities remain fixed throughout the computation, so there is an opportunity to *compile* the network into model-specific inference code that carries out just the inference computations needed for that specific network. (In case this sounds familiar, it is the same idea used in the compilation of logic programs in Chapter 9.) For example, suppose we want to Gibbs-sample the *Earthquake* variable in the burglary network of Figure 13.2. According to the GIBBS-ASK algorithm in Figure 13.20, we need to perform the following computation:

set the value of *Earthquake* in **x** by sampling from $\mathbf{P}(Earthquake \mid mb(Earthquake))$

where the latter distribution is computed according to Equation (13.10), repeated here:

$$P(x_i \mid mb(X_i)) = \alpha P(x_i \mid parents(X_i)) \prod_{Y_j \in Children(X_i)} P(y_j \mid parents(Y_j)) .$$

This computation, in turn, requires looking up the parents and children of *Earthquake* in the Bayes net structure; looking up their current values; using those values to index into the corresponding CPTs (which also have to be found from the Bayes net); and multiplying together all the appropriate rows from those CPTs to form a new distribution from which to sample. Finally, as noted on page 454, the sampling step itself has to construct the cumulative version of the discrete distribution and then find the value therein that corresponds to a random number sampled from $[0,1]$.

If, instead, we compile the network, we obtain model-specific sampling code for the *Earthquake* variable that looks like this:

```
$r \leftarrow$ a uniform random sample from $[0,1]$
if $Alarm = true$
    then if $Burglary = true$
        then return $[r < 0.0020212]$
        else return $[r < 0.36755]$
    else if $Burglary = true$
        then return $[r < 0.0016672]$
        else return $[r < 0.0014222]$
```

Here, Bayes net variables *Alarm*, *Burglary*, and so on become ordinary program variables with values that comprise the current state of the Markov chain. The numerical threshold expressions evaluate to *true* or *false* and represent the precomputed Gibbs distributions for each combination of values in the Markov blanket of *Earthquake*. The code is not especially pretty—typically, it will be roughly as large as the Bayes net itself—but it is incredibly efficient. Compared to GIBBS-ASK, the compiled code will typically be 2–3 orders of magnitude faster. It can perform tens of millions of sampling steps per second on an ordinary laptop, and its speed is limited largely by the cost of generating random numbers.

13.5 Causal Networks

We have discussed several advantages of keeping node ordering in Bayes nets compatible with the direction of causation. In particular, we noted the ease with which conditional probabilities can be assessed if such ordering is maintained, as well as the compactness of the resultant network structure. We noted however that, in principle, any node ordering permits a consistent construction of the network to represent the joint distribution function. This was demonstrated in Figure 13.3, where changing the node ordering produced networks that were bushier and a lot less natural than the original network in Figure 13.2 but enabled us, nevertheless, to represent the same distribution on all variables.

Causal network

This section describes **causal networks**, a restricted class of Bayesian networks that forbids all but causally compatible orderings. We will explore how to construct such networks, what is gained by such construction, and how to leverage this gain in decision-making tasks.

Consider the simplest Bayesian network imaginable, a single arrow, $\mathit{Fire} \rightarrow \mathit{Smoke}$. It tells us that variables *Fire* and *Smoke* may be dependent, so one needs to specify the prior $P(\mathit{Fire})$ and the conditional probability $P(\mathit{Smoke}\,|\,\mathit{Fire})$ in order to specify the joint distribution $P(\mathit{Fire}, \mathit{Smoke})$. However, this distribution can be represented equally well by the reverse arrow $\mathit{Fire} \leftarrow \mathit{Smoke}$, using the appropriate $P(\mathit{Smoke})$ and $P(\mathit{Fire}\,|\,\mathit{Smoke})$ computed from Bayes' rule. The idea that these two networks are equivalent, hence convey the same information, evokes discomfort and even resistance in most people. How could they convey the same information when we know that *Fire* causes *Smoke* and not the other way around?

In other words, we know from our experience and scientific understanding that clearing the smoke would not stop the fire and extinguishing the fire will stop the smoke. We expect therefore to represent this asymmetry through the directionality of the arrow between them. But if arrow reversal only makes things equivalent, how can we hope to represent this important information formally?

Causal Bayesian networks, sometimes called Causal Diagrams, were devised to permit us to represent causal asymmetries and to leverage the asymmetries towards reasoning with causal information. The idea is to decide on arrow directionality by considerations that go beyond probabilistic dependence and invoke a totally different type of judgment. Instead of asking an expert whether *Smoke* and *Fire* are probabilistically dependent, as we do in ordinary Bayesian networks, we now ask which responds to which, *Smoke* to *Fire* or *Fire* to *Smoke*?

This may sound a bit mystical, but it can be made precise through the notion of "assignment," similar to the assignment operator in programming languages. If nature assigns a value to *Smoke* on the basis of what nature learns about *Fire*, we draw an arrow from *Fire* to *Smoke*. More importantly, if we judge that nature assigns *Fire* a truth value that depends on other variables, not *Smoke*, we refrain from drawing the arrow $\mathit{Fire} \leftarrow \mathit{Smoke}$. In other words, the value x_i of each variable X_i is determined by an equation $x_i = f_i(\mathit{OtherVariables})$, and an arrow $X_j \rightarrow X_i$ is drawn if and only if X_j is one of the arguments of f_i.

Structural equation

The equation $x_i = f_i(\cdot)$ is called a **structural equation**, because it describes a stable mechanism in nature which, unlike the probabilities that quantify a Bayesian network, remains invariant to measurements and local changes in the environment.

To appreciate this stability to local changes, consider Figure 13.23(a), which depicts a slightly modified version of the lawn sprinkler story of Figure 13.15. To represent a disabled sprinkler, for example, we simply delete from the network all links incident to the *Sprinkler* node. To represent a lawn covered by a tent, we simply delete the arrow $\mathit{Rain} \rightarrow \mathit{WetGrass}$. Any local reconfiguration of the mechanisms in the environment can thus be translated, with only minor modification, into an isomorphic reconfiguration of the network topology. A much more elaborate transformation would be required had the network been constructed contrary to causal ordering. This local stability is particularly important for representing actions or interventions, our next topic of discussion.

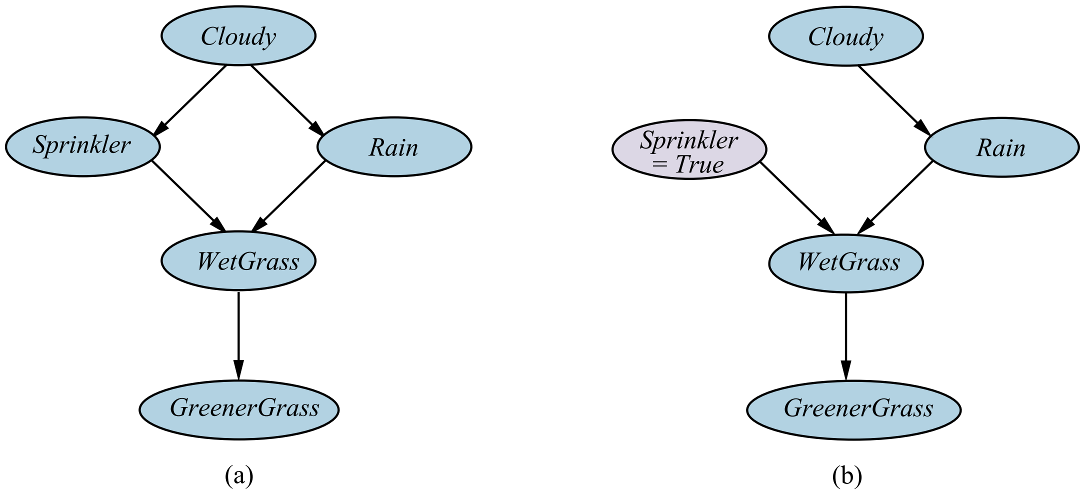


Figure 13.23 (a) A causal Bayesian network representing cause–effect relations among five variables. (b) The network after performing the action "turn *Sprinkler* on."

13.5.1 Representing actions: The *do*-operator

Consider again the *Sprinkler* story of Figure 13.23(a). According to the standard semantics of Bayes nets, the joint distribution of the five variables is given by a product of five conditional distributions:

$$P(c,r,s,w,g) = P(c)\,P(r\,|\,c)\,P(s\,|\,c)\,P(w\,|\,r,s)\,P(g\,|\,w) \tag{13.14}$$

where we have abbreviated each variable name by its first letter. As a system of structural equations, the model looks like this:

$$\begin{aligned} C &= f_C(U_C) \\ R &= f_R(C, U_R) \\ S &= f_S(C, U_S) \\ W &= f_W(R, S, U_W) \\ G &= f_G(W, U_G) \end{aligned} \tag{13.15}$$

where, without loss of generality, f_C can be the identity function. The U-variables in these equations represent **unmodeled variables**, also called **error terms** or **disturbances**, that perturb the functional relationship between each variable and its parents. For example, U_W may represent another potential source of wetness, in addition to *Sprinkler* and *Rain*—perhaps *MorningDew* or *FirefightingHelicopter*.

Unmodeled variable

If all the U-variables are mutually independent random variables with suitably chosen priors, the joint distribution in Equation (13.14) can be represented exactly by the structural equations in Equation (13.15). Thus, a system of stochastic relationships can be captured by a system of deterministic relationships, each of which is affected by an exogenous disturbance. However, the system of structural equations gives us more than that: it allows us to predict how *interventions* will affect the operation of the system and hence the observable consequences of those interventions. This is not possible given just the joint distribution.

For example, suppose we *turn the sprinkler on*—that is, if we (who are, by definition, not part of the causal processes described by the model) *intervene* to impose the condition $Sprinkler = true$. In the notation of the **do-calculus**, which is a key part of the theory of causal networks, this is written as $do(Sprinkler = true)$. Once done, this means that the sprinkler variable is no longer dependent on whether it's a cloudy day. We therefore delete the equation $S = f_S(C, U_S)$ from the system of structural equations and replace it with $S = true$, giving us

Do-calculus

$$\begin{aligned} C &= f_C(U_C) \\ R &= f_R(C, U_R) \\ S &= true \\ W &= f_W(R, S, U_W) \\ G &= f_G(W, U_G)\,. \end{aligned} \tag{13.16}$$

From these equations, we obtain the new joint distribution for the remaining variables conditioned on $do(Sprinkler = true)$:

$$P(c, r, w, g \mid do(S = true)) = P(c)\ P(r \mid c)\ P(w \mid r, s = true)\ P(g \mid w) \tag{13.17}$$

This corresponds to the "mutilated" network in Figure 13.23(b). From Equation (13.17), we see that the only variables whose probabilities change are *WetGrass* and *GreenerGrass*, that is, the descendants of the manipulated variable *Sprinkler*.

Note the difference between conditioning on the *action* $do(Sprinkler = true)$ in the original network and conditioning on the *observation* $Sprinkler = true$. The original network tells us that the sprinkler is less likely to be on when the weather is cloudy, so if we *observe* the sprinkler to be on, that reduces the probability that the weather is cloudy. But common sense tells us that if we (operating from outside the world, so to speak) reach in and turn on the sprinkler, that doesn't affect the weather or provide new information about what the weather is like that day. As shown in Figure 13.23(b), intervening breaks the normal causal link between the weather and the sprinkler. This prevents any influence flowing backward from *Sprinkler* to *Cloudy*. Thus, conditioning on $do(Sprinkler = true)$ in the original graph is equivalent to conditioning on $Sprinkler = true$ in the mutilated graph.

A similar approach can be taken to analyze the effect of $do(X_j = x_{jk})$ in a general causal network with variables $X_1, \ldots, X_n$. The network corresponds to a joint distribution defined in the usual way (see Equation (13.2)):

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))\,. \tag{13.18}$$

After applying $do(X_j = x_{jk})$, the new joint distribution $P_{x_{jk}}$ simply omits the factor for X_j:

$$P_{x_{jk}}(x_1, \ldots, x_n) = \begin{cases} \prod_{i \neq j} P(x_i \mid parents(X_i)) = \frac{P(x_1, \ldots, x_n)}{P(x_j \mid parents(X_j))} & \text{if } x_j = x_{jk} \\ 0 & \text{if } x_j \neq x_{jk} \end{cases} \tag{13.19}$$

This follows from the fact that setting X_j to a particular value x_{jk} corresponds to deleting the equation $X_j = f_j(Parents(X_j), U_j)$ from the system of structural equations and replacing it with $X_j = x_{jk}$. With a bit more algebraic manipulation, one can derive a formula for the effect of setting variable X_j on any other variable X_i:

$$\begin{aligned} P(X_i = x_i \mid do(X_j = x_{jk})) &= P_{x_{jk}}(X_i = x_i) \\ &= \sum_{parents(X_j)} P(x_i \mid x_{jk}, parents(X_j)) P(parents(X_j))\,. \end{aligned} \tag{13.20}$$

The probability terms in the sum are obtained by computation on the original network, by any of the standard inference algorithms. This equation is known as an **adjustment formula**; it is a probability-weighted average of the influence of X_j and its parents on X_i, where the weights are the priors on the parent values. The effects of intervening on multiple variables can be computed by imagining that the individual interventions happen in sequence, each one in turn deleting the causal influences on a variable and yielding a new, mutilated model.

Adjustment formula

13.5.2 The back-door criterion

The ability to predict the effect of any intervention is a remarkable result, but it does require accurate knowledge of the necessary conditional distributions in the model, particularly $P(x_j \mid parents(X_j))$. In many real-world settings, however, this is too much to ask. For example, we know that "genetic factors" play a role in obesity, but we do not know which genes play a role or the precise nature of their effects. Even in the simple story of Mary's sprinkler decisions (Figure 13.15, which also applies in Figure 13.23(a)), we might know that she checks the weather before deciding whether to turn on the sprinkler, but we might not know *how* she makes her decision.

The specific reason this is problematic in this instance is that we would like to predict the effect of turning on the sprinkler on a downstream variable such as *GreenerGrass*, but the adjustment formula (Equation (13.20)) must take into account not only the direct route from *Sprinkler*, but also the "back door" route via *Cloudy* and *Rain*. If we knew the value of *Rain*, this back-door path would be blocked—which suggests that there might be a way to write an adjustment formula that conditions on *Rain* instead of *Cloudy*. And indeed this is possible:

$$P(g \mid do(S = true)) = \sum_r P(g \mid S = true, r)P(r) \tag{13.21}$$

In general, if we wish to find the effect of $do(X_j = x_{jk})$ on a variable X_i, the **back-door criterion** allows us to write an adjustment formula that conditions on any set of variables $\mathbf{Z}$ that closes the back door, so to speak. In more technical language, we want a set $\mathbf{Z}$ such that X_i is conditionally independent of $Parents(X_j)$ given X_j and $\mathbf{Z}$. This is a straightforward application of d-separation (see page 437).

Back-door criterion

The back-door criterion is a basic building block for a theory of causal reasoning that has emerged in the past two decades. It provides a way to argue against a century of statistical dogma asserting that only a **randomized controlled trial** can provide causal information. The theory has provided conceptual tools and algorithms for causal analysis in a wide range of non-experimental and quasi-experimental settings; for computing probabilities on counterfactual statements ("if this had happened instead, what would the probability have been?"); for determining when findings in one population can be transferred to another; and for handling all forms of missing data when learning probability models.

Randomized controlled trial

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 joint probability distribution over its variables. The probability of any given assignment to all the variables is defined as the product of the corresponding entries in the local conditional distributions. A Bayesian network is often exponentially smaller than an explicitly enumerated joint distribution.
- Many conditional distributions can be represented compactly by canonical families of distributions. **Hybrid Bayesian networks**, which include both discrete and continuous variables, use a variety of canonical distributions.
- Inference in Bayesian networks means computing the probability distribution of a set of query variables, given a set of evidence variables. Exact inference algorithms, such as **variable elimination**, evaluate sums of products of conditional probabilities as efficiently as possible.
- In **polytrees** (singly connected networks), exact inference takes time linear in the size of the network. In the general case, the problem is intractable.
- Random sampling 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.
- Whereas Bayes nets capture probabilistic influences, **causal networks** capture causal relationships and allow prediction of the effects of interventions as well as observations.

Bibliographical and Historical Notes

The use of networks to represent probabilistic information began early in the 20th century, with the work of Sewall Wright on the probabilistic analysis of genetic inheritance and animal growth factors (Wright, 1921, 1934). I. J. Good (1961), in collaboration with Alan Turing, developed probabilistic representations and Bayesian inference methods that could be regarded as a forerunner of modern Bayesian networks—although the paper is not often cited in this context.[7] 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 15), but only enumeration was used for evaluation. Judea Pearl developed the message-passing method for inference in tree networks (Pearl, 1982a) and polytree networks (Kim and Pearl, 1983) and explained the importance of causal rather than diagnostic probability models. The first expert system using Bayesian networks was CONVINCE (Kim, 1983).

As chronicled in Chapter 1, the mid-1980s saw a boom in rule-based expert systems, which incorporated ad hoc methods for handling uncertainty. Probability was considered both impractical and "cognitively implausible" as a basis for reasoning. Peter Cheeseman's (1985)

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

pugnacious "In Defense of Probability" and his later article "An Inquiry into Computer Understanding" (Cheeseman, 1988, with commentaries) helped to turn the tables.

The resurgence of probability depended mainly, however, on Pearl's development of Bayesian networks and the broad development of a probabilistic approach to AI as outlined in his book, *Probabilistic Reasoning in Intelligent Systems* (Pearl, 1988). The book covered both representational issues, including conditional independence relationships and the d-separation criterion, and algorithmic approaches. Geiger *et al.* (1990a) and Tian *et al.* (1998) presented key computational results on efficient detection of d-separation.

Eugene Charniak helped present Pearl's ideas to AI researchers with a popular article, "Bayesian networks without tears"[8] (1991), and book (1993). The book by Dean and Wellman (1991) also helped introduce Bayesian networks to AI researchers. Shachter (1998) presented a simplified way to determine d-separation called the "Bayes-ball" algorithm.

As applications of Bayes nets were developed, researchers found it necessary to go beyond the basic model of discrete variables with CPTs. For example, the CPCS system (Pradhan *et al.*, 1994), a Bayesian network for internal medicine with 448 nodes and 906 links, made extensive use of the noisy logical operators proposed by Good (1961). Boutilier *et al.* (1996) analyzed the algorithmic benefits of context-specific independence. 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.

Hybrid networks with both discrete and continuous variables were investigated by Lauritzen and Wermuth (1989) and implemented in the cHUGIN system (Olesen, 1993). Further analysis of linear–Gaussian models, with connections to many other models used in statistics, appears in Roweis and Ghahramani (1999); Lerner (2002) provides a very thorough discussion of their use in hybrid Bayes nets. The probit distribution is usually attributed to Gaddum (1933) and Bliss (1934), although it had been discovered several times in the 19th century. Bliss's work was expanded considerably by Finney (1947). The probit has been used widely for modeling discrete choice phenomena and can be extended to handle more than two choices (Daganzo, 1979). The expit (inverse logit) model was introduced by Berkson (1944); initially much derided, it eventually became more popular than the probit model. Bishop (1995) gives a simple justification for its use.

Early applications of Bayes nets in medicine included the MUNIN system for diagnosing neuromuscular disorders (Andersen *et al.*, 1989) and the PATHFINDER system for pathology (Heckerman, 1991). Applications in engineering include the Electric Power Research Institute's work on monitoring power generators (Morjaria *et al.*, 1995), NASA's work on displaying time-critical information at Mission Control in Houston (Horvitz and Barry, 1995), and the general field of **network tomography**, which aims to infer unobserved local properties of nodes and links in the Internet from observations of end-to-end message performance (Castro *et al.*, 2004). Perhaps the most widely used Bayesian network systems have been the diagnosis-and-repair modules (e.g., the Printer Wizard) in Microsoft Windows (Breese and Heckerman, 1996) and the Office Assistant in Microsoft Office (Horvitz *et al.*, 1998).

Another important application area is biology: the mathematical models used to analyze genetic inheritance in family trees (so-called **pedigree analysis**) are in fact a special form

Pedigree analysis

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

of Bayesian networks. Exact inference algorithms for pedigree analysis, resembling variable elimination, were developed in the 1970s (Cannings *et al.*, 1978). Bayesian networks have been used for identifying human genes by reference to mouse genes (Zhang *et al.*, 2003), inferring cellular networks (Friedman, 2004), genetic linkage analysis to locate disease-related genes (Silberstein *et al.*, 2013), and many other tasks in bioinformatics. We could go on, but instead we'll refer you to Pourret *et al.* (2008), a 400-page guide to applications of Bayesian networks. Published applications over the last decade run into the tens of thousands, ranging from dentistry to global climate models.

Judea Pearl (1985), in the first paper to use the term "Bayesian networks," briefly described an inference algorithm for general networks based on the cutset conditioning idea introduced in Chapter 5. Independently, Ross Shachter (1986), working in the influence diagram community, developed a complete algorithm based on goal-directed reduction of the network using posterior-preserving transformations.

Pearl (1986) developed a clustering algorithm for exact inference in general Bayesian networks, utilizing a conversion to a directed polytree of clusters in which message passing was used to achieve consistency over variables shared between clusters. A similar approach, developed by the statisticians David Spiegelhalter and Steffen Lauritzen (Lauritzen and Spiegelhalter, 1988), is based on conversion to an undirected form of graphical model called a **Markov network**. This approach is implemented in the HUGIN system, an efficient and widely used tool for uncertain reasoning (Andersen *et al.*, 1989).

The basic idea of variable elimination—that repeated computations within the overall sum-of-products expression can be avoided by caching—appeared in the symbolic probabilistic inference (SPI) algorithm (Shachter *et al.*, 1990). The elimination algorithm we describe is closest to that developed by Zhang and Poole (1994). Criteria for pruning irrelevant variables were developed by Geiger *et al.* (1990b) and by Lauritzen *et al.* (1990); the criterion we give is a simple special case of these. Dechter (1999) shows how the variable elimination idea is essentially identical to **nonserial dynamic programming** (Bertele and Brioschi, 1972).

Nonserial dynamic programming

This connects Bayesian network algorithms to related methods for solving CSPs and gives a direct measure of the complexity of exact inference in terms of the tree width of the network. Preventing the exponential growth in the size of factors in variable elimination can be done by dropping variables from large factors (Dechter and Rish, 2003); it is also possible to bound the error introduced thereby (Wexler and Meek, 2009). Alternatively, factors can be compressed by representing them using algebraic decision diagrams instead of tables (Gogate and Domingos, 2011).

Exact methods based on recursive enumeration (see Figure 13.11) combined with caching include the recursive conditioning algorithm (Darwiche, 2001), the value elimination algorithm (Bacchus *et al.*, 2003), and AND–OR search (Dechter and Mateescu, 2007). The method of weighted model counting (Sang *et al.*, 2005; Chavira and Darwiche, 2008) is usually based on a DPLL-style SAT solver (see Figure 7.17 on page 252). As such, it is also performing a recursive enumeration of variable assignments with caching, so the approach is in fact quite similar. All three of these algorithms can implement a complete range of space/time tradeoffs. Because they consider variable assignments, the algorithms can easily take advantage of determinism and context-specific independence in the model. They can also be modified to use an efficient linear-time algorithm whenever the partial assignment makes the remaining network a polytree. (This is a version of the method of **cutset conditioning**, which was described for CSPs in Chapter 5.) For exact inference in large models, where the space requirements of clustering and variable elimination become enormous, these recursive algorithms are often the most practical approach.

Most probable explanation

There are other important inference tasks in Bayes nets besides computing marginal probabilities. The **most probable explanation** or MPE is the most likely assignment to the nonevidence variables given the evidence. (MPE is a special case of MAP—maximum a posteriori—inference, which asks for the most likely assignment to a *subset* of nonevidence variables given the evidence.) For such problems, many different algorithms have been developed, some related to shortest-path or AND–OR search algorithms; for a summary, see Marinescu and Dechter (2009).

The first result on the complexity of inference in Bayes nets is due to Cooper (1990), who showed that the general problem of computing marginals in Bayesian networks is NP-hard; as noted in the chapter, this can be strengthened to #P-hardness through a reduction from counting satisfying assignments (Roth, 1996). This also implies the NP-hardness of approximate inference (Dagum and Luby, 1993); however, for the case where probabilities can be bounded away from 0 and 1, a form of likelihood weighting converges in (randomized) polynomial time (Dagum and Luby, 1997). Shimony (1994) showed that finding the most probable explanation is NP-complete—intractable, but somewhat easier than computing marginals—while Park and Darwiche (2004) provide a thorough complexity analysis of MAP computation, showing that it falls into the class of $\mathrm{NP}^{\mathrm{PP}}$-complete problems—that is, somewhat harder than computing marginals.

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 dating back at least to Buffon's needle (1777); it was first applied to Bayesian networks by Max Henrion (1988), who called it **logic sampling**. Importance sampling was invented originally for applications in physics (Kahn, 1950a, 1950b) and applied to Bayes net inference by Fung and Chang (1989) (who called the algorithm "evidence weighting") and by Shachter and Peot (1989).

In statistics, **adaptive sampling** has been applied to all sorts of Monte Carlo algorithms to speed up convergence. The basic idea is to adapt the distribution from which samples are generated, based on the outcome from previous samples. Gilks and Wild (1992) developed adaptive rejection sampling, while adaptive importance sampling appears to have originated independently in physics (Lepage, 1978), civil engineering (Karamchandani *et al.*, 1989), statistics (Oh and Berger, 1992), and computer graphics (Veach and Guibas, 1995). Cheng and Druzdzel (2000) describe an adaptive version of importance sampling applied to Bayes net inference. More recently, Le *et al.* (2017) have demonstrated the use of deep learning systems to produce proposal distributions that speed up importance sampling by many orders of magnitude.

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. Hastings (1970) introduced the accept/reject step that is an integral part of what we now call the Metropolis–Hastings algorithm. The Gibbs sampler was devised by Geman and Geman (1984) for inference in undirected Markov networks. The application of Gibbs sampling to Bayesian networks is due to Pearl (1987). The papers collected by Gilks *et al.* (1996) cover both theory and applications of MCMC.

Since the mid-1990s, MCMC has become the workhorse of Bayesian statistics and statistical computation in many other disciplines including physics and biology. The *Handbook of Markov Chain Monte Carlo* (Brooks *et al.*, 2011) covers many aspects of this literature. The BUGS package (Gilks *et al.*, 1994) was an early and influential system for Bayes net modeling and inference using Gibbs sampling. STAN (named after Stanislaw Ulam, an originator of Monte Carlo methods in physics) is a more recent system that uses Hamiltonian Monte Carlo inference (Carpenter *et al.*, 2017).

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 accurate lower-bound approximations for sigmoid networks with the use of mean-field methods. Jaakkola and Jordan (1996) extended the methodology to obtain both lower and upper bounds. Since these early papers, variational methods have been applied to many specific families of models. The remarkable paper by Wainwright and Jordan (2008) provides a unifying theoretical analysis of the literature on variational methods.

A second important family of approximation algorithms is based on Pearl's polytree message-passing algorithm (1982a). This algorithm can be applied to general "loopy" networks, as suggested by Pearl (1988). The results might be incorrect, or the algorithm might fail to terminate, but in many cases, the values obtained are close to the true values. Little attention was paid to this so-called **loopy belief propagation** approach until McEliece *et al.* (1998) observed that it is 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.

Loopy belief propagation
Turbo decoding

The implication of these observations is if loopy BP is both fast and accurate on the very large and very highly connected networks used for decoding, it might therefore be useful more generally. Theoretical support for these findings, including convergence proofs for some special cases, was provided by Weiss (2000b), Weiss and Freeman (2001), and Yedidia *et al.* (2005), drawing on connections to ideas from statistical physics.

Theories of causal inference going beyond randomized controlled trials were proposed by Rubin (1974) and Robins (1986), but these ideas remained both obscure and controversial until Judea Pearl developed and presented a fully articulated theory of causality based on causal networks (Pearl, 2000). Peters *et al.* (2017) further develop the theory, with an emphasis on learning. A more recent work, *The Book of Why* (Pearl and McKenzie, 2018), provides a less mathematical but more readable and wide-ranging introduction.

Uncertain reasoning in AI has not always been based on probability theory. As noted in Chapter 12, early probabilistic systems fell out of favor in the early 1970s, leaving a partial vacuum to be filled by alternative methods. These included rule-based expert systems, Dempster–Shafer theory, and (to some extent) fuzzy logic.[9]

Rule-based approaches to uncertainty hoped to build on the success of logical rule-based systems, but add a sort of "fudge factor"—more politely called a **certainty factor**—to each rule to accommodate uncertainty. The first such system was MYCIN (Shortliffe, 1976), a medical expert system for bacterial infections. 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. 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. The basic mathematical properties that allow *chains* of reasoning in logic simply do not hold for probability.

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. Such an approach might alleviate the difficulty of specifying probabilities exactly. Later work by Glenn Shafer (1976) led to the Dempster–Shafer theory's being viewed as a competing approach to probability. Pearl (1988) and Ruspini *et al.* (1992) analyze the relationship between the Dempster–Shafer theory and standard probability theory. In many cases, probability theory does not require probabilities to be specified exactly: we can express uncertainty about probability values as (second-order) probability distributions, as explained in Chapter 21.

Fuzzy sets were developed by Lotfi Zadeh (1965) in response to the perceived difficulty of providing exact inputs to intelligent systems. A fuzzy set is one in which membership is a matter of degree. **Fuzzy logic** is a method for reasoning with logical expressions describing membership in fuzzy sets. **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. The text by Zimmermann (2001) provides a thorough introduction to fuzzy set theory; papers on fuzzy applications are collected in Zimmermann (1999).

Fuzzy logic has often been perceived incorrectly as a direct competitor to probability theory, whereas in fact it addresses a different set of issues: rather than considering uncertainty about the truth of well-defined propositions, fuzzy logic handles **vagueness** in the mapping from terms in a symbolic theory to an actual world. Vagueness is a real issue in any application of logic, probability, or indeed standard mathematical models to reality. Even a variable as impeccable as the mass of the Earth turns out, on inspection, to vary with time as meteorites and molecules come and go. It is also imprecise—does it include the atmosphere? If so, to what height? In some cases, further elaboration of the model can reduce vagueness, but fuzzy logic takes vagueness as a given and develops a theory around it.

[9] A fourth approach, **default reasoning**, treats conclusions not as "believed to a certain degree," but as "believed until a better reason is found to believe something else." It is covered in Chapter 10.

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).

Many AI researchers in the 1970s rejected probability because the numerical calculations that probability theory was thought to require were not apparent to introspection and presumed an unrealistic level of precision in our uncertain knowledge. The development of **qualitative probabilistic networks** (Wellman, 1990a) provided a purely qualitative abstraction of Bayesian networks, using the notion of positive and negative influences between variables. Wellman shows that in many cases such information is sufficient for optimal decision making without the need for the precise specification of probability values. Goldszmidt and Pearl (1996) take a similar approach. Work by Darwiche and 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.

Several excellent texts (Jensen, 2007; Darwiche, 2009; Koller and Friedman, 2009; Korb and Nicholson, 2010; Dechter, 2019) provide thorough treatments of the topics we have covered in this chapter. New research on probabilistic reasoning appears both in mainstream AI journals, such as *Artificial Intelligence* and the *Journal of AI Research*, and in more specialized journals, such as the *International Journal of Approximate Reasoning*. Many papers on graphical models, which include Bayesian networks, appear in statistical journals. The proceedings of the conferences on Uncertainty in Artificial Intelligence (UAI), Neural Information Processing Systems (NeurIPS), and Artificial Intelligence and Statistics (AISTATS) are good sources for current research.