14.5 APPROXIMATE INFERENCE IN BAYESIAN NETWORKS

MONTE CARLO

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

14.5.1 Direct sampling methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CONSISTENT

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

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

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

Rejection sampling in Bayesian networks

REJECTION SAMPLING

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

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

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

From Equation (14.6), this becomes

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

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

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

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

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

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

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

Likelihood weighting

LIKELIHOOD WEIGHTING

IMPORTANCE SAMPLING

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

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

  for j = 1 to N do
      x ← PRIOR-SAMPLE(bn)
      if x is consistent with e then
          N[x] ← N[x]+1 where x is the value of X in x
  return NORMALIZE(N)
```

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

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

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

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

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

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

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

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
    N, the total number of samples to be generated
  local variables: **W**, a vector of weighted counts for each value of X, initially zero

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

function WEIGHTED-SAMPLE(bn, **e**) **returns** an event and a weight

  $w \leftarrow 1$; **x** $\leftarrow$ an event with n elements initialized from **e**
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    if X_i is an evidence variable with value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else **x**$[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return **x**, w

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

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

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

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

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

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

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

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

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

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

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

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

Hence, likelihood weighting returns consistent estimates.

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

14.5.2 Inference by Markov chain simulation

MARKOV CHAIN MONTE CARLO

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

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

GIBBS SAMPLING

Gibbs sampling in Bayesian networks

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

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

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

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

Why Gibbs sampling works

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

TRANSITION PROBABILITY

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

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

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

MARKOV CHAIN

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

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

STATIONARY DISTRIBUTION

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

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

ERGODIC

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

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

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

DETAILED BALANCE

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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