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

14.5 APPROXIMATE INFERENCE IN BAYESIAN NETWORKS

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

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 (heads, tails) and a prior distribution $P(Coin) = (0.5, 0.5)$. 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, Ram, WetGrass].

1. Sample from $P(Cloudy) = (0.5, 0.5)$, value is true.
2. Sample from $P(Sprinkler | Cloudy = true) = 0.9$, value is false.
3. Sample from $P(Rain | Cloudy = true) = (0.8, 0.2)$, value is true.
4. Sample from $P(WetGrass | Sprinkler = false, Rain = true) = (0, 0.5)$, value is true.

In this case, PRIOR-SAMPLE returns the event [true, false, true, true].
function \texttt{PRIOR-SAMPLE}(bn) returns an event sampled from the prior specified by \( bn \)
inputs: \( bn \), a Bayesian network specifying joint distribution \( P(X_1, \ldots, X_n) \)
x \( n \times n \) an event with \( n \) elements
foreach variable \( X_i \) in \( X_1, \ldots, X_n \) do
\( x[i] \) \( \leftarrow \) a random sample from \( P(X, \text{parents}(X_i)) \)
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 \texttt{PRIOR-SAMPLE} generates samples from the prior joint distribution specified by the network. Let \( S_{PS} (x_1, \ldots, x_n) \) be the probability that a specific event is generated by the \texttt{PRIOR-SAMPLE} algorithm. Just looking at the sampling process, we have
\[
S_{PS} (x_1, \ldots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))
\]
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:
\[
\lim_{N \to \infty} \frac{N_{PS} (x_1, \ldots, x_n)}{N} = S_{PS} (x_1, \ldots, x_n) = P(x_1, \ldots, x_n).
\]
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 32.49% 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 \( m < n \), as follows:
\[
P(x_1, \ldots, x_m) \approx \frac{N_{PS} (x_1, \ldots, x_m)}{N}.
\]
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