The operation of the three schemes will be illustrated with a simple example borrowed from Spiegelhalter [1986], originally by Cooper [1984]:

Metastatic cancer is a possible cause of a brain tumor and is also an explanation for increased total serum calcium. In turn, either of these could explain a patient falling into a coma. Severe headache is also possibly associated with a brain tumor.

Figure 4.23 shows the Bayesian network representing these relationships. As in the preceeding sections, we use uppercase letters to represent propositional variables and lowercase letters for their associated propositions. For example, $C \in \{1, 0\}$ represents the dichotomy between having a brain tumor and not having one. $+c$ stands for the assertion $C = 1$ or "Brain tumor is present," and $\neg c$ stands for the negation of $+c$, i.e., $C = 0$.

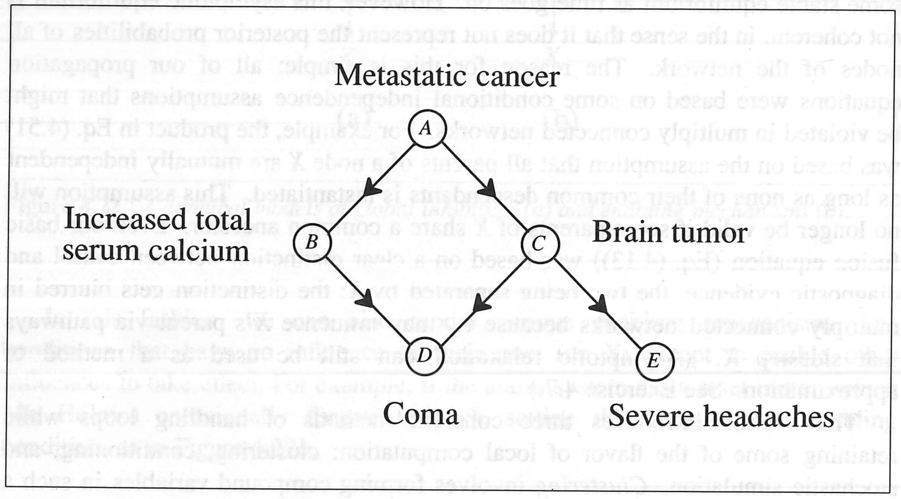

Figure 4.23. *A Bayesian network describing causal influences among five variables.*

Table 1 expresses the influences in terms of conditional probability distributions. Each variable is characterized by a *link matrix,* specifying the probability distribution of that variable given the state of its parents.† The root variable, having no parent, is characterized by its prior distribution.

† The probabilities are for illustration purposes only, and are not meant to realistically reflect current medical knowledge. Additionally, the variable "Coma" should be interpreted to mean "Lapsing occasionally into coma"; otherwise it would preclude headaches.

Table 1.

$P(a)$:	$P(+a) = .20$	
$P(b \mid a)$:	$P(+b \mid +a) = .80$	$P(+b \mid \neg a) = .20$
$P(c \mid a)$:	$P(+c \mid +a) = .20$	$P(+c \mid \neg a) = .05$
$P(d \mid b, c)$:	$P(+d \mid +b, +c) = .80$	$P(+d \mid \neg b, +c) = .80$
	$P(+d \mid +b, \neg c) = .80$	$P(+d \mid \neg b, \neg c) = .05$
$P(e \mid c)$:	$P(+e \mid +c) = .80$	$P(+e \mid \neg c) = .60$

Given this information, our task is to compute the posterior probability of every proposition in the system, given that a patient is suffering from severe headaches ($+e$) but has not fallen into a coma ($\neg d$), i.e., $e = \{E = 1, D = 0\}$.

4.4.1 Clustering Methods

A straightforward way of handling the network of Figure 4.23 would be to collapse B and C into a single node representing the compound variable $Z = \{B, C\}$ with the values

$$z \in \{(+b, +c), (\neg b, +c), (+b, \neg c), (\neg b, \neg c)\} . \qquad \textbf{(4.65)}$$

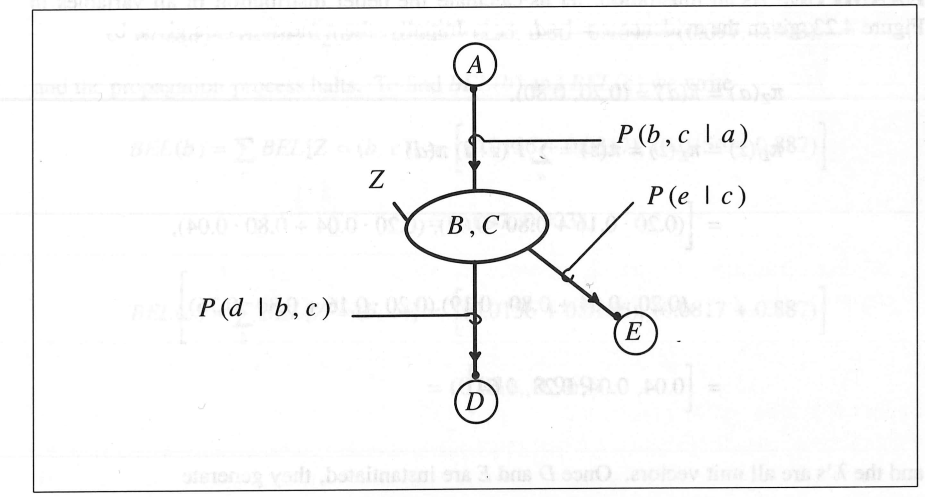

Figure 4.24. *Clustering B and C turns the network of Figure 4.23 into a tree.*

This results in the tree structure shown in Figure 4.24. Since the cardinality of variable Z is 4, the matrices on all three links must have either four rows or four

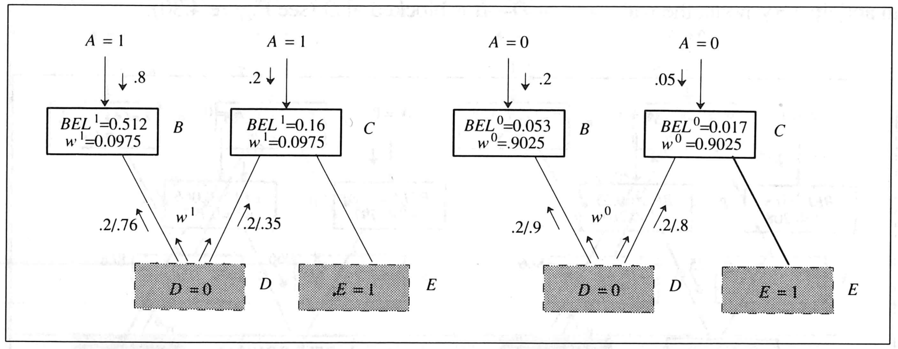

Figure 4.31. *Updated beliefs, messages, and weights after observing $E = 1$ and $D = 0$. Beliefs are computed by the combination $BEL = w^1\ BEL^1 + w^0\ BEL^0$.*

At this point, all belief distributions can be computed at their corresponding nodes, as in Figure 4.31:

$$BEL(b) = w^1_{E,D}\ BEL^1(b) + w^0_{E,D}\ BEL^0(b),$$
$$BEL(c) = w^1_{E,D}\ BEL^1(c) + w^0_{E,D}\ BEL^0(c),$$

where

$$BEL^1(b) = \alpha^1\ \pi^1(b)\ \lambda^1_D(b) = \alpha^1(0.8 \cdot 0.2,\ 0.2 \cdot 0.76) = (0.512,\ 0.488),$$
$$BEL^0(b) = \alpha^0\ \pi^0(b)\ \lambda^0_D(b) = \alpha^0(0.2 \cdot 0.2,\ 0.8 \cdot 0.9) = (0.053,\ 0.947),$$
$$BEL^1(c) = \alpha^1\ \pi^1(c)\ \lambda^1_D(c)\ \lambda_E(c) = \alpha^1(0.2 \cdot 0.2 \cdot 0.80,\ 0.8 \cdot 0.35 \cdot 0.60) = (0.16,\ 0.84),$$
$$BEL^0(c) = \alpha^0\ \pi^0(c)\ \lambda^0_D(c)\ \lambda_E(c) = \alpha^0(0.05 \cdot 0.2 \cdot 0.80,\ 0.95 \cdot 0.8 \cdot 0.60) = (0.017,\ 0.983).$$

These yield

$$BEL(b) = (0.096,\ 0.904),$$
$$BEL(c) = (0.031,\ 0.964).$$

$BEL(a)$, of course, is equal to the current mixing weight $w_{E,D} = (0.0975,\ 0.9025)$.

4.4.3 *Stochastic Simulation*

Stochastic simulation is a method of computing probabilities by counting how frequently events occur in a series of simulation runs. If a causal model of a domain is available, the model can be used to generate random samples of

hypothetical scenarios that are likely to develop in the domain. The probability of any event or combination of events can then be computed by counting the percentage of samples in which the event is true.

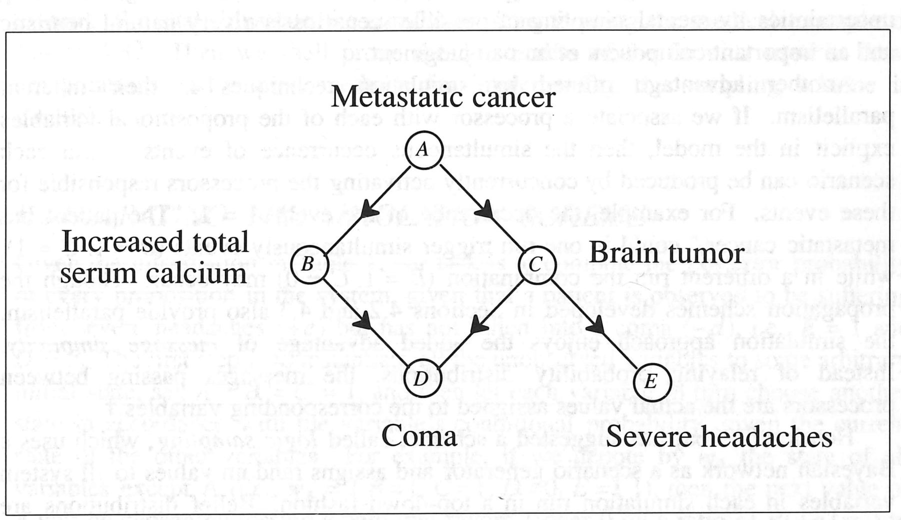

Figure 4.32. *The Bayesian network used to demonstrate stochastic simulation (same as in Figure 4.23).*

For example, in the causal model of Figure 4.32, we can generate hypothetical samples of patients by the following procedure: We draw a random value a_1 for A, using the probability $P(a)$. Given a_1, we draw random values b_1 and c_1 for the variables B and C, using the probabilities $P(b \mid a_1)$ and $P(c \mid a_1)$, respectively. Given b_1 and c_1, we draw random values d_1 and e_1 for D and E, using $P(d \mid b_1, c_1)$ and $P(e \mid c_1)$, respectively. The combination of values $(a_1, b_1, c_1, d_1, e_1)$ represents one sample of a patient scenario. The process now repeats from A down to D and E, each run generating a quintuple that represents one patient.

Stochastic simulation shows considerable potential as a probabilistic inference engine that combines evidence correctly but is computationally tractable. Unlike numerical schemes, the computational effort is unaffected by the presence of dependencies within the causal model; simulating the occurrence of an event given the states of its causes requires the same computational effort regardless of whether the causes are correlated. In our example above, simulating the event D given the states of events B and C was straightforward, even though B and C are correlated (via A). Thus, the presence of loops in the network does not affect the computation.

Stochastic simulation carries a special appeal for AI researchers in that it develops probabilistic reasoning as a direct extension of deterministic logical inference. It represents probabilities explicitly as "frequencies" in a sample of truth values, and these values, unlike numerical probabilities, can be derived by familiar theorem-proving techniques and combined by standard logical connectives. Nor is the technique foreign to human reasoning; assessing uncertainties by mental sampling of possible scenarios is a very natural heuristic and an important component of human judgment.

Another advantage offered by simulation techniques is their inherent parallelism. If we associate a processor with each of the propositional variables explicit in the model, then the simultaneous occurrence of events within each scenario can be produced by concurrently activating the processors responsible for these events. For example, the occurrence of the event $A = 1$, "The patient has metastatic cancer," could in one run trigger simultaneously events $(B = 1, C = 1)$, while in a different run the combination $(B = 1, C = 0)$ may occur. Though the propagation schemes developed in Sections 4.2 and 4.3 also provide parallelism, the simulation approach enjoys the added advantage of *message simplicity*. Instead of relaying probability distributions, the messages passing between processors are the actual values assigned to the corresponding variables.†

Henrion [1986a] has suggested a scheme, called *logic sampling*, which uses a Bayesian network as a scenario generator and assigns random values to all system variables in each simulation run in a top-down fashion. Belief distributions are calculated by averaging the frequency of events over those cases in which the evidence variables agree with the data observed. This scheme retains the merits of causal modeling in that it conducts the simulation along the flow of causation, so that each step can be given a conceptually meaningful interpretation. Since the simulation proceeds only forward in time, however, there is no way to account for evidence known to have occurred (e.g., $\neg d$, $+e$) until the variables corresponding to these observations are sampled. If they match the observed data, the run is counted; otherwise, it must be discarded. The result is that the scheme requires too many simulation runs. In cases comprising large numbers of observations (e.g., 20), all but a small fraction (e.g., 10^{-6}) of the simulations may be discarded, especially when a rare combination of data occurs.

A better way to account for the evidence would be to permanently clamp the evidence variables to the values observed, and then conduct a stochastic simulation on the clamped network. The question that remains is how to propagate the random values coherently through the network, now that boundary conditions are imposed on both the top and bottom nodes, i.e., on premises as well as consequences.

This section describes such a propagation method, involving a two-phase cycle: local numerical computation followed by logical sampling. The first

† This conforms to the connectionist paradigm of reasoning [Rumelhart and McClelland 1986], in which processors are presumed to communicate by merely passing their levels of activity.

phase involves computing, for some variable X, the conditional distribution given the states of all its neighboring variables. The second phase involves sampling the computed distribution and instantiating X to the value selected by the sampling. The cycle then repeats itself by sequentially scanning all the variables in the system. We shall illustrate the simulation scheme using the medical example of Figure 4.32. Then we shall prove the correctness of the formula used in these computations and discuss methods for implementing the sampling scheme in parallel.

ILLUSTRATING THE SIMULATION SCHEME

Given the information in Table 1, our task is to compute the posterior probability of every proposition in the system, given that a patient is observed to be suffering from severe headaches $(+e)$ but has not fallen into a coma $(\neg d)$, i.e., $E = 1$ and $D = 0$. The first step is to instantiate all the unobserved variables to some arbitrary initial state, say $A = B = C = 1$, and then let each variable in turn choose another state in accordance with the variable's conditional probability, given the current state of the other variables. For example, if we denote by w_A the state of all variables except A (i.e., $w_A = \{ B=1, C=1, D=0, E=1 \}$), then the next value of A will be chosen by tossing a coin that favors 1 over 0 by a ratio of $P(+a \mid w_A)$ to $P(\neg a \mid w_A)$.

In the next subsection, we shall show that $P(x \mid w_X)$, the distribution of each variable X conditioned on the values w_X of all other variables in the system, can be calculated by purely local computations. It is given as the product of the link matrix of X and the link matrices of its children:

$$P(a \mid w_A) = P(a \mid b, c, d, e) = \alpha\, P(a)\, P(b \mid a)\, P(c \mid a), \tag{4.69a}$$

$$P(b \mid w_B) = P(b \mid a, c, d, e) = \alpha\, P(b \mid a)\, P(d \mid b, c), \tag{4.69b}$$

$$P(c \mid w_C) = P(c \mid a, b, d, e) = \alpha\, P(c \mid a)\, P(d \mid b, c)\, P(e \mid c), \tag{4.69c}$$

where the α's are normalizing constants that make the respective probabilities sum to unity. The probabilities associated with D and E are not needed because these variables are assumed to be fixed at $D = 0$ and $E = 1$. Note that a variable X can determine its transition probability $P(x \mid w_X)$ by inspecting only *neighboring* variables, i.e., those belonging to X's *Markov blanket* (see Section 3.3.1, Corollary 6). For example, A must inspect only B and C, while B must inspect only A, C, and D.

EXAMPLE: For demonstration purposes, we will activate the variables sequentially, in the order A, B, C, acknowledging that any other schedule would be equally adequate.

ACTIVATING A

Step 1: Node A inspects its children B and C; finding both at 1, it computes (using Eq. (4.69*a*))

$$\begin{aligned} P(A=1\,|\,w_A) = P(A=1\,|\,B=1, C=1) &= \alpha\, P(a)\, P(+b\,|\,+a)\, P(+c\,|\,+a) \\ &= \alpha \times 0.20 \times 0.80 \times 0.20 \\ &= \alpha \times 0.032, \end{aligned}$$

$$\begin{aligned} P(A=0\,|\,w_A) = P(A=0\,|\,B{=}1, C{=}1) &= \alpha\, P(\neg a)\, P(+b\,|\,\neg a)\, P(+c\,|\,\neg a) \\ &= \alpha \times 0.80 \times 0.20 \times 0.05 \\ &= \alpha \times 0.008, \end{aligned}$$

$$\alpha = [0.032 + 0.008]^{-1} = 25,$$

yielding

$$P(A=1\,|\,w_A) = 25 \times 0.032 = 0.80,$$
$$P(A=0\,|\,w_A) = 25 \times 0.008 = 0.20.$$

Step 2: Node A consults a random number generator that issues 1s with 80% probability and 0s with 20% probability. Assuming the value sampled is 1, A adopts this value, and control shifts to node B.

ACTIVATING B

Step 1: Node B inspects its neighbors; finding them with values $A=1$, $C=1$, and $D=0$, it computes (using Eq. (4.69*b*))

$$\frac{P(B=1\,|\,w_B)}{P(B=0\,|\,w_B)} = \frac{P(B=1\,|\,A=1, C=1, D=0)}{P(B=0\,|\,A=1, C=1, D=0)} = \frac{\alpha\, P(+b\,|\,+a)\, P(\neg d\,|\,+b, +c)}{\alpha\, P(\neg b\,|\,+a)\, P(\neg d\,|\,\neg b, +c)}$$
$$= \frac{0.80 \times (1-0.80)}{(1-0.80)(1-0.80)} = \frac{4}{1}.$$

Step 2:

As A did in its turn, B samples a random number generator favoring 1 by a 4 to 1 ratio. Assuming, this time, that the value sampled is 0, B adopts the value 0 and gives control to C.

ACTIVATING C

Step 1: The neighbors of C are at the state

$$w_C = \{A=1, B=0, D=0, E=1\}.$$

Therefore, from Eq. (4.69*c*):

$$\frac{P(+c\,|\,w_C)}{P(\neg c\,|\,w_C)} = \frac{P(+c\,|\,+a)\, P(\neg d\,|\,\neg b, +c)\, P(+e\,|\,+c)}{P(\neg c\,|\,+a)\, P(\neg d\,|\,\neg b, \neg c)\, P(+e\,|\,\neg c)}$$
$$= \frac{0.20 \times (1-0.80)\; 0.80}{(1-0.20)(1-0.05)\; 0.60} = \frac{1}{14.25}$$

Step 2: C samples a random number generator favoring 0 by a 14.25 to 1 ratio. Assuming the value 0 is sampled, C adopts the value 0 and gives control to A.

ANSWERING QUERIES

The cycle now repeats itself in the order A, B, C until a query is posted, e.g., "What is the posterior distribution of A?" Such a query can be answered by computing the percentage of times A registered the value 1 or by taking the average of the conditional probabilities $P(A{=}1\,|\,w_A)$ computed by A. The latter method usually yields faster convergence.

To illustrate, the value of $P(A=1\,|\,w_A)$ computed in the next activation of A would be

$$\begin{aligned} P(A=1\,|\,B=0, C=0) &= \alpha\, P(+a)\, (\neg b\,|\,+a)\, P(\neg c\,|\,+a) \\ &= \alpha\ 0.20\,(1-0.80)\,(1-0.20) \\ &= \alpha\ 0.032, \end{aligned}$$

$$\begin{aligned} P(A=0\,|\,B=0, C=0) &= \alpha\, P(\neg a)\, P(\neg b\,|\,\neg a)\, P(\neg c\,|\,\neg a) \\ &= \alpha\ 0.80\,(1-0.20)\,(1-0.05) \\ &= \alpha\ 0.608, \end{aligned}$$

$$\alpha = (0.032 + 0.608)^{-1} = 1.5625,$$

$$P(A{=}1\,|\,B{=}0, C{=}0) = 0.05,$$
$$P(A{=}0\,|\,B{=}0, C{=}0) = 0.95.$$

If a query "$P(+a\,|\,\neg d, +e) = ?$" arrives at this point, A samples the computed distribution (i.e., $P(a) = 0.05$) and upon selecting a value 0 provides the estimate

$$\hat{P}(+a\,|\,\neg d, +e) = \frac{1+0}{2} = 0.5.$$

The second method gives

$$\hat{P}(+a\,|\,\neg d, +e) = \frac{0.80 + 0.05}{2} = 0.425\,.$$

The exact value of $P(+a \mid \neg d, +e)$ happens to be 0.097 (see the calculations in Sections 4.4.1 and 4.4.2); it takes over 100 runs for $\hat{P}$ to come within 1% of this value.

The convergence of $\hat{P}$ to the correct value of the posterior probability is guaranteed, under certain conditions, by a theorem of Feller [1950] regarding the existence of a limiting distribution for Markov chains. In each simulation run the system's configuration changes from state i to state j, and the change is governed by the transition probability $P(x \mid w_X)$ of the activated variable. The essence of Feller's theorem is this: if for any pair (i, j) of configurations there is a positive probability of reaching j from i in a finite number of transitions, then regardless of the initial configuration, the probability that the system will be found at a given state approaches a limit, which is determined by the *stationarity* condition

$$P(j) = \sum_i P(i)\, P(j \mid i), \qquad \textbf{(4.70)}$$

where $P(j \mid i)$ is the probability of reaching state j from state i in one transition. In our case, the reachability condition is guaranteed if all link probabilities are positive, because every configuration then has a positive probability of being realized in one run (over all variables). Thus, the fact that the transition probabilities $P(x \mid w_X)$ satisfy Eq. (4.70) relative to distribution P is sufficient to guarantee that the asymptotic probability distribution—and hence $\hat{P}$—will converge to P. In other words, as time progresses the system is guaranteed to reach a steady state, in the sense that regardless of the initial instantiation, the probability that the system will enter any global state w is given by the joint distribution $P(w_X)$ specified by the link matrices. The case where some link probabilities are zero corresponds to *reducible* Markov chains and limits the applicability of stochastic simulation schemes (see the concluding subsection).

This simulation scheme can also be used to find the *most likely interpretation* of the observed data, i.e., a joint assignment w^* of values to all variables that has the highest posterior probability of all possible assignments, given the evidence. This will be discussed in Chapter 5.

JUSTIFYING THE COMPUTATIONS

We shall now prove the correctness of the product formula (Eq. (4.69)) used for computing the transition probabilities $P(x \mid w_X)$. Clearly, the conditional distribution of X given the state of all remaining variables is sensitive not to every variable in the system but only to those in the *neighborhood* of X, namely, the variables that if known would render X independent of all other variables in the system. Such a neighborhood, called a *Markov blanket* $\boldsymbol{B}_X$ of X, was identified in Section 3.3.1 (Corollary 6) as comprised of three types of neighbors: direct

parents, direct successors, and all direct parents of direct successors. In Figure 4.32, for example, the Markov blankets for each variable are given as follows:

$$\boldsymbol{B}_A = \{B, C\}, \quad \boldsymbol{B}_B = \{A, C, D\},$$
$$\boldsymbol{B}_C = \{A, B, D, E\}, \quad \boldsymbol{B}_D = \{B, C\}, \quad \boldsymbol{B}_E = \{C\}.$$

Yet, replacing $P(x \mid w_X)$ with $P(x \mid \boldsymbol{b}_X)$ will not be very helpful unless the latter can be easily computed from the link matrices surrounding X. Next we shall show that $P(x \mid w_X)$ consists of a product of $m+1$ link matrices, where m is the number of X's children.

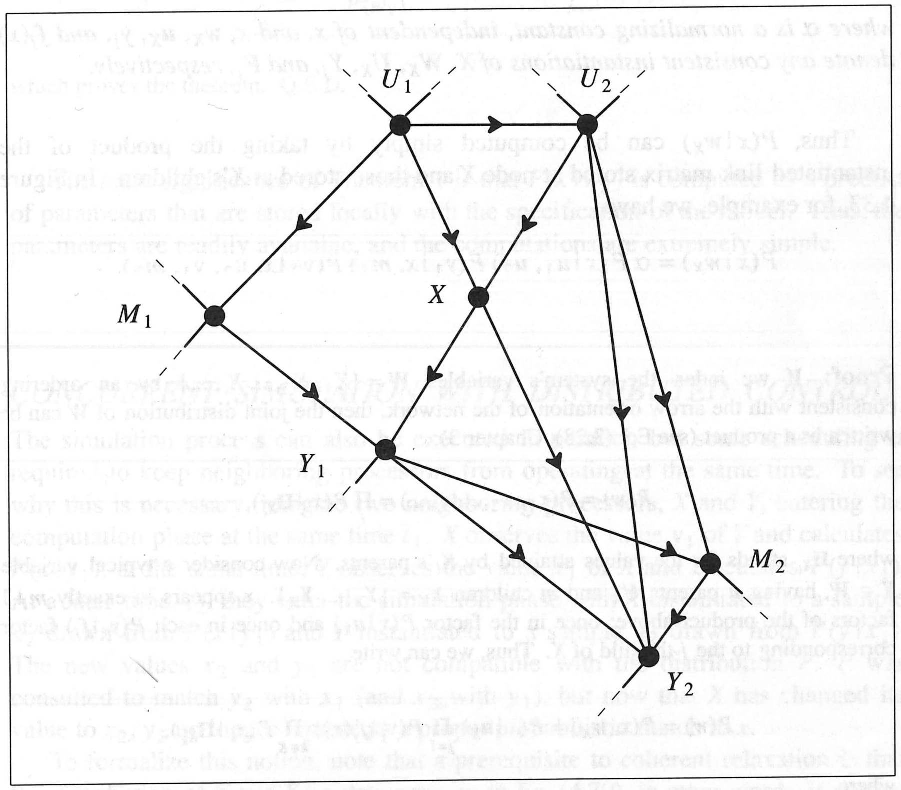

Figure 4.33. *The Markov neighborhood of X, including parents (U_1, U_2), children (Y_1, Y_2), and mates (M_1, M_2).*

Consider a typical neighborhood of variable X in some Bayesian network, as shown in Figure 4.33: Define the following set of variables:

1. X's parents, $\boldsymbol{U}_X = \{U_1, ..., U_n\}$.
2. X's children, $\boldsymbol{Y}_X = \{Y_1, ..., Y_m\}$.
3. $\boldsymbol{F}_j$, the set of parents of Y_j.
4. $\boldsymbol{W}_X = \boldsymbol{W} - X$, the set of all variables except X.

THEOREM 1: *The probability distribution of each variable X in the network, conditioned on the state of all other variables, is given by the product*

$$P(x \mid \boldsymbol{w}_X) = \alpha \, P(x \mid \boldsymbol{u}_X) \prod_j P[y_j \mid \boldsymbol{f}_j(x)], \qquad \textbf{(4.71)}$$

where α is a normalizing constant, independent of x, and x, $\boldsymbol{w}_X$, $\boldsymbol{u}_X$, y_j, and $\boldsymbol{f}_j(x)$ denote any consistent instantiations of X, $\boldsymbol{W}_X$, $\boldsymbol{U}_X$, Y_j, and $\boldsymbol{F}_j$, respectively.

Thus, $P(x \mid \boldsymbol{w}_X)$ can be computed simply by taking the product of the instantiated link matrix stored at node X and those stored at X's children. In Figure 4.33, for example, we have

$$P(x \mid \boldsymbol{w}_X) = \alpha \, P(x \mid u_1, u_2) \, P(y_1 \mid x, m_1) \, P(y_2 \mid x, u_2, y_1, m_2).$$

Proof: If we index the system's variables $\boldsymbol{W} = \{X_1, X_2, ..., X_i, ...,\}$ by an ordering consistent with the arrow orientation of the network, then the joint distribution of $\boldsymbol{W}$ can be written as a product (see Eq. (3.28), Chapter 3):

$$P(w) = P(x_1, x_2, ..., x_i, ...) = \prod_i P(x_i \mid \Pi_{X_i}),$$

where Π_{X_i} stands for the values attained by X_i's parents. Now consider a typical variable $X \in \boldsymbol{W}$, having n parents $\boldsymbol{U}_X$ and m children $\boldsymbol{Y}_X = \{Y_1, ..., Y_m\}$. x appears in exactly $m+1$ factors of the product above; once in the factor $P(x \mid \boldsymbol{u}_X)$ and once in each $P(y_j \mid \boldsymbol{f}_j)$ factor corresponding to the j-th child of X. Thus, we can write

$$P(w) = P(x, \boldsymbol{w}_X) = P(x \mid \boldsymbol{u}_X) \prod_{j=1}^{m} P(y_j \mid \boldsymbol{f}_j(x)) \prod_{k \in K} P(x_k \mid \Pi_{X_k}),$$

where

$$K = \{k \colon X_k \in \boldsymbol{W}_X - \boldsymbol{Y}_X\}.$$

Since x does not appear in the rightmost product (over k), this product can be regarded as a constant α' relative to x, and we can write

$$P(x, \boldsymbol{w}_X) = \alpha' P(x \mid \boldsymbol{u}_X) \prod_j (y_j \mid \boldsymbol{f}_j(x)).$$

Moreover, since

$$P(\boldsymbol{w}_X) = \sum_x P(x, \boldsymbol{w}_X)$$

is also a constant relative to x, we have

$$P(x \mid \boldsymbol{w}_X) = \frac{P(x, \boldsymbol{w}_X)}{P(\boldsymbol{w}_X)} = \alpha \, P(x \mid \boldsymbol{u}_X) \prod_j P(y_j \mid \boldsymbol{f}_j(x)),$$

which proves the theorem. Q.E.D.

The main significance of Theorem 1 is that $P(x \mid \boldsymbol{w}_X)$ is computed as a product of parameters that are stored locally with the specification of the model. Thus, the parameters are readily available, and the computations are extremely simple.

CONCURRENT SIMULATION WITH DISTRIBUTED CONTROL

The simulation process can also be executed in parallel, but some scheduling is required to keep neighboring processors from operating at the same time. To see why this is necessary, imagine two neighboring processors, X and Y, entering the computation phase at the same time t_1. X observes the value y_1 of Y and calculates $P(x \mid y_1)$; at the same time, Y observes the value x_1 of X and calculates $P(y \mid x_1)$. At a later time, t_2, they enter the simulation phase with X instantiated to a sample x_2 drawn from $P(x \mid y_1)$ and Y instantiated to a sample y_2 drawn from $P(y \mid x_1)$. The new values x_2 and y_2 are not compatible with the distribution P. P was consulted to match y_2 with x_1 (and x_2 with y_1), but now that X has changed its value to x_2, y_2 no longer represents a proper probabilistic match to x.

To formalize this notion, note that a prerequisite to coherent relaxation is that the distribution of X and Y be stationary, as in Eq. (4.70). In other words, if at time t_1, X and Y are distributed by $P(x, y)$, then the values of X and Y at time t_2 must also be distributed by $P(x, y)$. This requirement is met when only one variable changes at any given time, because then we can write (assuming Y is the changing variable)

$$P(X_2 = x,\ Y_2 = y) = \sum_{x'y'} P(X_2 = x, Y_2 = y \mid X_1 = x', Y_1 = y')\, P(x', y')$$

$$= P(Y_1 = y \mid X_1 = x)\, P(X_1 = x)$$

$$= P(X_1 = x, Y_1 = y) = P(x, y), \tag{4.72}$$

which implies stationary distribution. If, however, X and Y change their values simultaneously, we have

$$P(X_2 = x, Y_2 = y) = P(Y_2 = y \mid X_2 = x)\, P((X_2 = x)$$

$$= \sum_{x'y'} P(X_2 = x, Y_2 = y \mid X_1 = x', Y_1 = y')\, P(x', y')$$

$$= \sum_{x'y'} P(X_1 = x \mid Y_1 = y')\, P(Y_1 = y \mid X_1 = x')\, P(x', y')$$

$$= \sum_{x'y'} \frac{P(x, y')}{P(y')} \frac{P(x', y)}{P(x')}\, P(x', y'), \tag{4.73}$$

which represents stationary distribution only in the pathological case where X_1 and Y_1 are independent.

This analysis, extended to the case of multiple variables, allows us to determine which variables can be activated simultaneously. Let the set of concurrently activated variables be $\mathbf{Z} = \{Z_1, Z_2, ..., Z_n\}$, and assume that each Z_i variable chooses a new value z_i' by sampling the distribution $P(z_i \mid s_i)$, where $\mathbf{S}_i$ is the subset of variables inspected by Z_i prior to switching. If $\mathbf{W_Z}$ stands for the set of unchanged variables, then under the requirement of stationary distribution,

$$P(z', w_Z) = P(z, w_Z)$$

or

$$P(z' \mid w_Z) = P(z \mid w_Z), \tag{4.74}$$

because $P(w_Z)$ remains unchanged in the transition.

Since the values z' of the $\mathbf{Z}$ variables are drawn independently from $P(z_i \mid s_i)$, Eq. (4.74) translates to

$$\prod_{i=1}^{n} P(Z_i = z_i \mid s_i) = P(z_1, z_2, ..., z_n \mid w_Z). \tag{4.75}$$

This requirement is satisfied whenever each $\mathbf{S}_i$ is a Markov blanket of Z_i,

$$P(z_i \mid s_i) = P(z_i \mid w_{Z_i}), \tag{4.76}$$

and, simultaneously, each $\mathbf{S}_i$ shields Z_i from all other Z's,

$$P(z_i \mid s_i) = P(z_i \mid s_i \bigwedge_{j \neq i} z_j) \qquad i = 1, 2, ..., n. \tag{4.77}$$

To meet both Eq. (4.76) and Eq. (4.77), it is clear that if $\mathbf{S}_i$ contains any of the Z_j's, then $\mathbf{S}_i - Z_j$ must also shield Z_i from all other Z's. However, if we assume that each $\mathbf{S}_i$ already is the smallest Markov blanket permitted by the network, we must conclude that no Z_j's should be a member of any of the $\mathbf{S}_i$'s. Thus, any set of variables licensed to be activated simultaneously must not contain a pair of variables belonging to the same Markov blanket.

A convenient way to enforce this requirement is to add dummy links between mates (i.e., nodes sharing a child), taking care that no two adjacent nodes in the augmented network are activated concurrently. The question now arises how to schedule the activation of the processors so that the following conditions hold:

1. No two adjacent processors are activated at the same time.
2. Every processor gets activated sufficiently often.
3. The activation commands are generated in distributed fashion, with no external supervision.

This problem is a version of the "dining philosophers" dilemma originally posed by Dijkstra [1972] and later solved independently by Gafni and Bertsekas [1981] and Chandy and Misra [1984]. The solution is a distributed control policy called *edge reversal*, involving the following steps:

1. The links of the network are assigned arbitrary acyclic arrow orientation. (This orientation bears no relation to the causal ordering governing the construction of Bayesian networks.)
2. Each processor inspects the orientation of the arrows on its incident links and waits until all arrows point inward, i.e., until the processor becomes a *sink*.
3. Once a processor becomes a sink, it is activated, and when it completes the computation, it reverses the direction of all its incident arrows (i.e., it becomes a *source*).

It is easily seen that no two neighbors can be activated at the same time. What is more remarkable about this edge reversal policy, however, is that no processor ever gets "deprived"; every processor fires at least once before the orientation returns to its initial state and the cycle repeats itself. This feature is important because it constitutes a necessary condition for the convergence of the entire process [Geman and Geman 1984].

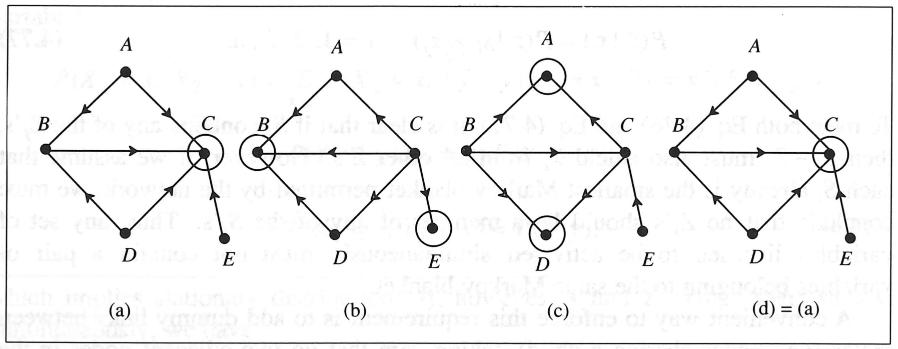

Figure 4.34. *Concurrent activation under the edge-reversal policy. Sinks fire and reverse their edges, thus ensuring that no two neighbors fire concurrently.*

Figure 4.34 applies this policy to the Bayesian network of Figure 4.32 by marking with circles the nodes activated at each step of the process. Initially, the dummy edge BC is added to designate these mates as neighbors, and the orientation of Figure 4.34*a* is assigned, where C is the only sink. Once C is activated, the arrows pointing to C are reversed (by C), whereupon B and E become sinks and fire. After three steps (Figure 4.34*d*), the orientation is back where it started, and the cycle repeats. Note that every processor fires once during the cycle and that we twice (Figures 4.34*b* and 4.34*c*) had two processors firing simultaneously. The problem of achieving maximum concurrency with edge reversal was analyzed by Barbosa [1986], who showed that the difference in the number of firings of any two nodes in the network cannot exceed a constant equal to the distance between them.

CONCLUSIONS

Stochastic simulation offers a viable inferencing technique for evidential reasoning tasks by virtue of its local and concurrent character. Although hundreds of runs may be necessary for achieving reasonable levels of accuracy, each run requires only $|V| + |E|$ computational steps, where $|V|$ is the number of vertices in the model and $|E|$ is the number of edges. Unlike purely numerical techniques, which sometimes entail exponential complexity, the length of computation is determined mainly by the required degree of accuracy, not by the dependencies embodied in the model. It is postulated, therefore, that stochastic simulation will be found practical in applications involving complex models with highly interdependent variables and in applications where "ballpark" estimates of probabilities will suffice.

The method has a drawback, however: the rate of convergence deteriorates when variables are constrained by functional dependencies. For example, if X and Y are a pair of adjacent variables constrained by equality, $X = Y$, then starting the

simulation in a state where $X = Y = 0$ will leave X and Y clamped to 0 forever. Likewise, starting with $X = Y = 1$ will leave them clamped to 1 even though states having $X = Y = 0$ may be more probable. If we permit the equality to be violated with small probability p, this clamping phenomenon disappears, but the rate of convergence still seems to be proportional to $1/p$ [Chin and Cooper 1987]. While the theory of Markov chains [Feller 1950] guarantees that the simulation counts obtained by stochastic simulation will converge to the correct posterior probabilities associated with each proposition, the theory requires that every conceivable state has a nonzero probability of occurring, and this requirement is violated under logical or functional constraints.

One way to speed up the convergence rate is to treat clusters of tightly constrained elements as singleton variables, conduct the simulation runs on the clustered network, and then compute the internal distribution of the elements within each cluster. If such clusters cannot be identified in advance, the stochastic simulation method should be restricted to Henrion's scheme of forward simulation, i.e., each variable reacts only to the state of its direct parents, ignoring the states of other neighbors. This will render the method robust to functional dependencies but may necessitate a large number of runs to match rare sets of observations. A method combining the merits of both the forward-driven and the neighborhood-driven simulation schemes has not yet been identified.

4.5 WHAT ELSE CAN BAYESIAN NETWORKS COMPUTE?

4.5.1 Answering Queries

Since a quantified Bayesian network represents a complete probabilistic model of the domain, and since one can easily use such a network to derive the joint probability distribution $P(x_1,..., x_n)$ for all variables involved, it is clear that the network contains sufficient information for computing answers to all queries regarding the variables $X_1,..., X_n$. In other words, if $q(x_1,..., x_n)$ stands for any Boolean combination of the propositions $X_1 = x_1, X_2 = x_2,..., X_n = x_n$, then an answer of the form $P(q)$ can always be computed from the joint distribution represented by the network. For example, if all variables are propositional, and q stands for the truth value of $[(X_2 = TRUE) \wedge (X_3 = TRUE)] \vee (X_6 = TRUE)$, then $P(q)$ can be calculated by summing $P(x_1,..., x_n)$ over all elementary events $(x_1 \wedge x_2,..., \wedge x_n)$ entailed by the event q. Our goal is to find an efficient network representation for that calculation.

So far, the propagation scheme developed in this chapter has been aimed at computing the belief function BEL for each node in the network, which amounts to