
Inference in Bayesian Networks

3.1 Introduction

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of query nodes, given values for some evidence nodes. This task is called **belief updating** or **probabilistic inference**. Inference in Bayesian networks is very flexible, as evidence can be entered about any node while beliefs in any other nodes are updated.

In this chapter we will cover the major classes of inference algorithms — exact and approximate — that have been developed over the past 20 years. As we will see, different algorithms are suited to different network structures and performance requirements. Networks that are simple chains merely require repeated application of Bayes' Theorem. Inference in simple tree structures can be done using local computations and message passing between nodes. When pairs of nodes in the BN are connected by multiple paths the inference algorithms become more complex. For some networks, exact inference becomes computationally infeasible, in which case approximate inference algorithms must be used. In general, both exact and approximate inference are theoretically computationally complex (specifically, NP hard). In practice, the speed of inference depends on factors such as the structure of the network, including how highly connected it is, how many undirected loops there are and the locations of evidence and query nodes.

Many inference algorithms have not seen the light of day beyond the research environment that produced them. Good exact and approximate inference algorithms are implemented in BN software, so knowledge engineers do not have to. Hence, our main focus is to characterize the main algorithms' computational performance to both enhance understanding of BN modeling and help the knowledge engineer assess which algorithm is best suited to the application. It is important that the belief updating is not merely a “black-box” process, as there are knowledge engineering issues that can only be resolved through an understanding of the inference process.

We will conclude the chapter with a discussion of how to use Bayesian networks for **causal modeling**, that is for reasoning about the effect of active interventions in the causal process being represented by the network. Such reasoning is important for hypothetical or counterfactual reasoning and for planning and control applications. Unfortunately, current BN tools do not explicitly support causal modeling; however, they can be used for such reasoning and we will describe how to do so.

3.2 Exact inference in chains

3.2.1 Two node network

We begin with the very simplest case, a two node network $X \rightarrow Y$.

If there is evidence about the parent node, say $X = x$, then the posterior probability (or belief) for Y , which we denote $Bel(Y)$, can be read straight from the value in CPT, $P(Y|X = x)$.

If there is evidence about the child node, say $Y = y$, then the inference task of updating the belief for X is done using a simple application of Bayes' Theorem.

$$\begin{aligned} Bel(X = x) &= P(X = x|Y = y) \\ &= \frac{P(Y = y|X = x)P(X = x)}{P(Y = y)} \\ &= \alpha P(x)\lambda(x) \end{aligned}$$

where

$$\alpha = \frac{1}{P(Y = y)}$$

$P(x)$ is the prior and $\lambda(x) = P(Y = y|X = x)$ is the **likelihood**. Note that we don't need to know the prior for the evidence. Since the beliefs for all the values of X must sum to one (due to the Total Probability Theorem 1.2), we can compute α , as a **normalizing constant**.

Example: $Flu \rightarrow HighTemp$

Suppose that we have this very simple model of flu causing a high temperature, with prior $P(Flu = T) = 0.05$, and CPT values $P(HighTemp = T|Flu = T) = 0.9$, $P(HighTemp = T|Flu = F) = 0.2$. If a person has a high temperature (i.e., the evidence available is $HighTemp = T$), the computation for this diagnostic reasoning is as follows.

$$\begin{aligned} Bel(Flu = T) &= \alpha P(Flu = T)\lambda(Flu = T) \\ &= \alpha \times 0.05 \times 0.9 \\ &= \alpha 0.045 \\ Bel(Flu = F) &= \alpha P(Flu = F)\lambda(Flu = F) \\ &= \alpha \times 0.95 \times 0.2 \\ &= \alpha 0.19 \end{aligned}$$

We can compute α via

$$Bel(Flu = T) + Bel(Flu = F) = 1 = \alpha 0.045 + \alpha 0.19$$

giving

$$\alpha = \frac{1}{0.19 + 0.045}$$

This allows us to finish the belief update:

$$\begin{aligned} Bel(Flu = T) &= \frac{0.045}{0.19 + 0.045} = 0.8085 \\ Bel(Flu = F) &= \frac{0.19}{0.19 + 0.045} = 0.1915 \end{aligned}$$

3.2.2 Three node chain

We can apply the same method when we have three nodes in a chain, $X \rightarrow Y \rightarrow Z$.

If we have evidence about the root node, $X = x$, updating in the same direction as the arcs involves the simple application of the chain rule (Theorem 1.3), using the independencies implied in the network.

$$Bel(Z) = P(Z|X = x) = \sum_{Y=y} P(Z|Y)P(Y|X = x)$$

If we have evidence about the leaf node, $Z = z$, the diagnostic inference to obtain $Bel(X)$ is done with the application of Bayes' Theorem and the chain rule.

$$\begin{aligned} Bel(X = x) &= P(X = x|Z = z) \\ &= \frac{P(Z = z|X = x)P(X = x)}{P(Z = z)} \\ &= \frac{\sum_{Y=y} P(Z = z|Y = y, X = x)P(Y = y|X = x)P(X = x)}{P(Z = z)} \\ &= \frac{\sum_{Y=y} P(Z = z|Y = y)P(Y = y|X = x)P(X = x)}{P(Z = z)} \quad (Z \perp\!\!\!\perp X|Y) \\ &= \alpha P(x)\lambda(x) \end{aligned}$$

where

$$\lambda(x) = P(Z = z|X = x) = \sum_{Y=y} P(Z = z|Y = y)P(Y = y|X = x)$$

Example: $Flu \rightarrow HighTemp \rightarrow HighTherm$

Here our flu example is extended by having an observation node *HighTherm* representing the reading given by a thermometer. The possible inaccuracy in the thermometer reading is represented by the following CPT entries:

- $P(HighTherm = T|HighTemp = T) = 0.95$ (so 5% chance of a false negative reading)
- $P(HighTherm = T|HighTemp = F) = 0.15$ (15% chance of a false positive reading).

Suppose that a high thermometer reading is taken, i.e., the evidence is $HighTherm = T$.

$$\begin{aligned}\lambda(Flu = T) &= P(HighTherm = T | HighTemp = T)P(HighTemp = T | Flu = T) \\ &\quad + P(HighTherm = T | HighTemp = F)P(HighTemp = F | Flu = T) \\ &= 0.95 \times 0.9 + 0.15 \times 0.1 \\ &= 0.855 + 0.015 = 0.87\end{aligned}$$

$$\begin{aligned}Bel(Flu = T) &= \alpha P(Flu = T) \lambda(Flu = T) \\ &= \alpha 0.05 \times 0.87 = \alpha 0.0435\end{aligned}$$

$$\begin{aligned}\lambda(Flu = F) &= P(HighTherm = T | HighTemp = T)P(HighTemp = T | Flu = F) \\ &\quad + P(HighTherm = T | HighTemp = F)P(HighTemp = F | Flu = F) \\ &= 0.95 \times 0.2 + 0.15 \times 0.8 \\ &= 0.19 + 0.12 = 0.31\end{aligned}$$

$$\begin{aligned}Bel(Flu = F) &= \alpha P(Flu = F) \lambda(Flu = F) \\ &= \alpha 0.95 \times 0.31 = \alpha 0.2945\end{aligned}$$

So,

$$\alpha = \frac{1}{0.0435 + 0.2945}$$

hence

$$\begin{aligned}Bel(Flu = T) &= \frac{0.0435}{0.0435 + 0.2945} = 0.1287 \\ Bel(Flu = F) &= \frac{0.2945}{0.0435 + 0.2945} = 0.8713 \quad (= 1 - Bel(Flu = T))\end{aligned}$$

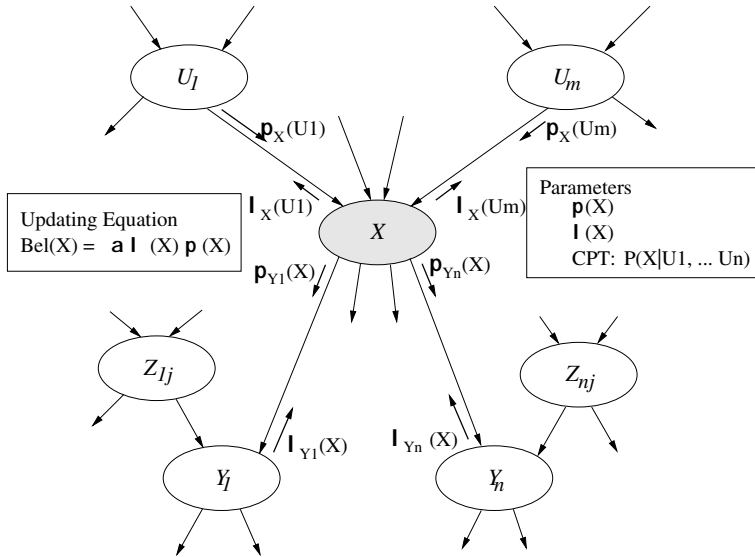
Clearly, inference in chains is straightforward, using application of Bayes' Theorem and the conditional independence assumptions represented in the chain. We will see that these are also the fundamental components of inference algorithms for more complex structures.

3.3 Exact inference in polytrees

Next we will look at how inference can be performed when the network is a simple structure called a **polytree** (or “**forest**”). Polytrees have at most one path between any pair of nodes; hence they are also referred to as **single-connected** networks.

Assume X is the query node, and there is some set of evidence nodes \mathbf{E} (not including X). The task is to update $Bel(X)$ by computing $P(X|\mathbf{E})$.

Figure 3.1 shows a diagram of a node X in a generic polytree, with all its connections to parents (the U_i), children (the Y_j), and the children's other parents (the Z_{ij}). The local belief updating for X must incorporate evidence from all other parts of the network. From this diagram we can see that evidence can be divided into:

**FIGURE 3.1**

A generic polytree showing how local belief updating of node X is achieved through incorporation of evidence through its parents (the U_i) and children (the Y_j). Also shown are the message passing parameters and messages.

- The **predictive support** for X , from evidence nodes connected to X through its parents, U_1, \dots, U_m ; and
- The **diagnostic support** for X , from evidence nodes connected to X through its children Y_1, \dots, Y_n .

3.3.1 Kim and Pearl's message passing algorithm

A “bare bones” description of Kim and Pearl's message passing algorithm appears below as Algorithm 3.1. The derivation of the major steps is beyond the scope of this text; suffice it to say, it involves the repeated application of Bayes' Theorem and use of the conditional independencies encoded in the network structure. Those details can be found elsewhere (e.g., [217, 237, 200]). Instead, we will present the major features of the algorithm, illustrating them by working through a simple example.

The parameters and messages used in this algorithm are also shown in Figure 3.1. The basic idea is that at each iteration of the algorithm, $Bel(X)$ is updated locally using three types of parameters — $\lambda(X)$, $\pi(X)$ and the CPTs (equation (3.1)). $\lambda(X)$ and $\pi(X)$ are computed using the π and λ messages received from X 's parents and children respectively. π and λ messages are also sent out from X so that its neighbors can perform updates. Let's look at some of the computations more closely.

ALGORITHM 3.1*Kim and Pearl's Message Passing Algorithm*

This algorithm requires the following three types of parameters to be maintained.

- *The current strength of the predictive support π contributed by each incoming link $U_i \rightarrow X$:*

$$\pi_X(U_i) = P(U_i | E_{U_i \setminus X})$$

where $E_{U_i \setminus X}$ is all evidence connected to U_i except via X .

- *The current strength of the diagnostic support λ contributed by each outgoing link $X \rightarrow Y_j$:*

$$\lambda_{Y_j}(X) = P(E_{Y_j \setminus X} | X)$$

where $E_{Y_j \setminus X}$ is all evidence connected to Y_j through its parents except via X .

- *The fixed CPT $P(X | U_i, \dots, U_n)$ (relating X to its immediate parents).*

These parameters are used to do local belief updating in the following three steps, which can be done in any order.

(Note: in this algorithm, x_i means the i th state of node X , while $u_1 \dots u_n$ is used to represent an instantiation of the parents of X , $U_1 \dots U_n$, in the situations where there is a summation of all possible instantiations.)

1. Belief updating.

Belief updating for a node X is activated by messages arriving from either children or parent nodes, indicating changes in their belief parameters.

When node X is activated, inspect $\pi_X(U_i)$ (messages from parents), $\lambda_{Y_j}(X)$ (messages from children). Apply with

$$Bel(x_i) = \alpha \lambda(x_i) \pi(x_i) \quad (3.1)$$

where,

$$\lambda(x_i) = \begin{cases} 1 & \text{if evidence is } X = x_i \\ 0 & \text{if evidence is for another } x_j \\ \prod_j \lambda_{Y_j}(x_i) & \text{otherwise} \end{cases} \quad (3.2)$$

$$\pi(x_i) = \sum_{u_1, \dots, u_n} P(x_i | u_1, \dots, u_n) \prod_i \pi_X(u_i) \quad (3.3)$$

and α is a normalizing constant rendering $\sum_{x_i} Bel(X = x_i) = 1$.

2. Bottom-up propagation.

Node X computes new λ messages to send to its parents.

$$\lambda_X(u_i) = \sum_{x_i} \lambda(x_i) \sum_{u_k : k \neq i} P(x_i | u_1, \dots, u_n) \prod_{k \neq i} \pi_X(u_k) \quad (3.4)$$

3. Top-down propagation.

Node X computes new π messages to send to its children.

$$\pi_{Y_j}(x_i) = \begin{cases} 1 & \text{if evidence value } x_i \text{ is entered} \\ 0 & \text{if evidence is for another value } x_j \\ \alpha [\prod_{k \neq j} \lambda_{Y_k}(x_i)] \sum_{u_1, \dots, u_n} P(x_i | u_1, \dots, u_n) \prod_i \pi_X(u_i) & \\ = \frac{\alpha \text{Bel}(x_i)}{\lambda_{Y_j}(x_i)} & \end{cases} \quad (3.5)$$

First, equation (3.2) shows how to compute the $\lambda(x_i)$ parameter. Evidence is entered through this parameter, so it is 1 if x_i is the evidence value, 0 if the evidence is for some other value x_j , and is the product of all the λ messages received from its children if there is no evidence entered for X . The $\pi(x_i)$ parameter (3.3) is the product of the CPT and the π messages from parents.

The λ message to one parent combines (i) information that has come from children via λ messages and been summarized in the $\lambda(X)$ parameter, (ii) the values in the CPT and (iii) any π messages that have been received from any other parents.

The $\pi_{Y_j}(x_i)$ message down to child Y_j is 1 if x_i is the evidence value and 0 if the evidence is for some other value x_j . If no evidence is entered for X , then it combines (i) information from children other than Y_j , (ii) the CPT and (iii) the π messages it has received from its parents.

The algorithm requires the following initializations (i.e., before any evidence is entered).

- Set all λ values, λ messages and π messages to 1.
- Root nodes: If node W has no parents, set $\pi(W)$ to the prior, $P(W)$.

The message passing algorithm can be used to compute the beliefs for all nodes in the network, even before any evidence is available.

When specific evidence $W = w_i$ is obtained, given that node W can take values $\{w_1, w_2, \dots, w_n\}$,

- Set $\lambda(W) = (0, 0, \dots, 0, 1, 0, \dots, 0)$ with the 1 at the i th position.

This π/λ notation used for the messages is that introduced by Kim and Pearl and can appear confusing at first. Note that the format for both types of messages is $\pi_{Child}(Parent)$ and $\lambda_{Child}(Parent)$. So,

- π messages are sent in the direction of the arc, from parent to child, hence the notation is $\pi_{Receiver}(Sender)$;
- λ messages are sent from child to parent, against the direction of the arc, hence the notation is $\lambda_{Sender}(Receiver)$.

Note also that π plays the role of prior and λ the likelihood in Bayes' Theorem.

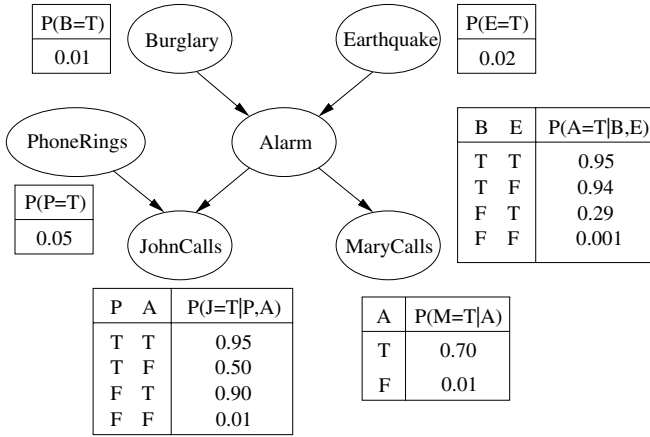


FIGURE 3.2
Extended earthquake BN.

3.3.2 Message passing example

Here, we extend the BN given in Figure 2.6 for Pearl's earthquake problem, by adding a node *PhoneRings* to represent explicitly the phone ringing that John sometimes confuses with the alarm. This extended BN is shown in Figure 3.2, with the parameters and messages used in the message passing algorithm shown in Figure 3.3.

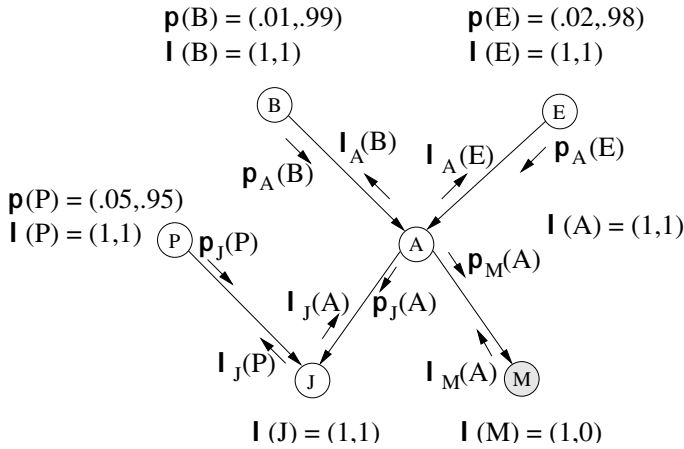
We will work through the message passing updating for the extended earthquake problem, first without evidence, then with evidence entered for node *M*. The data propagation messages are shown in Figure 3.4. The updating and propagation sequencing presented here are not necessarily those that a particular algorithm would produce, but rather the most efficient sequencing to do the belief updating in the minimum number of steps.

First, before any evidence is entered, various parameters are initialized: $\pi(B)$, $\pi(E)$, $\pi(P)$, from the priors, and $\lambda(J)$, $\lambda(M)$ with (1,1) as leaf nodes without evidence.

During this propagation before evidence, all of the diagnostic messages (the λ messages) will be the unit vector. We can see from the updating equations that we only multiply by these messages, so multiplying by the unit vector will not change any other parameters or messages. So we will not show the unit vector λ message propagation at this stage.

Using the belief updating equation (3.1), $Bel(B)$, $Bel(E)$ and $Bel(P)$ are computed, while sending new messages $\pi_A(B)$, $\pi_A(E)$ and $\pi_J(P)$ are computed using (3.5) and sent out.

Node *A* has received all its π messages from its parents, so it can update $Bel(A)$ and compute its own π messages to pass down to its children *J* and *M*. At this stage, we *could* update $Bel(J)$, as it has just received a π message from *P*; however, it has yet to receive a π message from *A*, so we won't do that this cycle.

**FIGURE 3.3**

Extended earthquake BN showing parameter initialization and π and λ messages for Kim and Pearl's message passing algorithm.

After the second message propagation phase, $Bel(J)$ and $Bel(M)$ can be computed, as all π messages have been received from their parents.

Next, we look at how evidence $M = T$, entered by setting $\lambda(M) = (1, 0)$, can be propagated through the network. First, the message $\lambda_M(A)$ is computed and sent up to A . $\lambda(A)$ and in turn $Bel(A)$ are then recomputed, and new messages sent to A 's parents via $\lambda_A(B)$ and $\lambda_A(E)$, and to its other child J via $\pi_J(A)$.

These new messages allow $Bel(B)$, $Bel(E)$ and $Bel(J)$ to be recomputed, and the final message $\lambda_J(P)$ computed and sent from J to P . The last computation is the updating of $Bel(P)$. Note that the minimum number of propagation steps for this evidence example is three, since that is the distance of the furthest node P from the evidence node M .

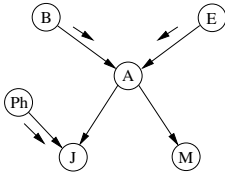
3.3.3 Algorithm features

All the computations in the message passing algorithm are local: the belief updating and new outgoing messages are all computed using incoming messages and the parameters. While this algorithm is efficient in a sense because of this locality property, and it lends itself to parallel, distributed implementations, we can see that there is a summation over all joint instantiations of the parent nodes, which is exponential in the number of parents. Thus, the algorithm is computationally infeasible when there are too many parents. And the longer the paths from the evidence node or nodes, the more cycles of data propagation must be performed to update all other nodes.

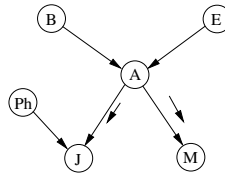
Note also that in the presentation of this algorithm, we have followed Pearl's presentation [217] and normalized all the messages. This is a computational overhead that is not strictly necessary, as all the normalizing can be done when computing the

PROPAGATION, NO EVIDENCE

PHASE 1

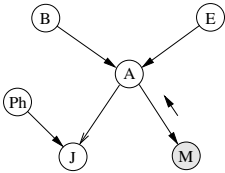


PHASE 2

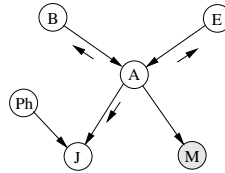


PROPAGATION, EVIDENCE for node M

PHASE 1



PHASE 2



PHASE 3

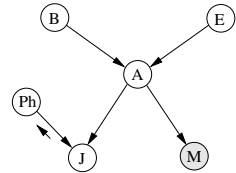


FIGURE 3.4

Message passing algorithm propagation stages for without evidence (above) and with evidence for node M (below).

marginals (i.e., when computing $Bel(x_i)$). The normalization constants, α , are the same for all the marginals, being the inverse of the probability $P(E)$, the computation of which is often useful for other purposes.

3.4 Inference with uncertain evidence

Thus far we have assumed that any evidence is a direct observation of the value of a variable that will result in the belief for a node being set to 1 for that value and 0 for all other values. This is the specific evidence described in §2.3.2, which is entered in the message passing algorithm as a vector with a 1 in the position of the evidence value and 0 in the other positions. Once specific evidence is entered for a node, the belief for that node is “clamped” and doesn’t change no matter what further information becomes available.

However, the inference algorithms should also be able to handle evidence that has uncertainty associated with it. In our earthquake example, suppose that after you get a call from your neighbor Mary saying she has heard your alarm going off, a colleague who is in your office at the time says he thinks he heard earlier on the radio that there was a minor earthquake in your area, but he is only 80% sure.

We introduced this notion very briefly in §2.3.2, where we mentioned that this sort of evidence is called “virtual” evidence or “likelihood” evidence. We deferred further

explanation of these terms until here, as they relate to how inference is performed.

Some of the major BN software packages (e.g., Netica and Hugin) provide the facility for adding likelihood evidence, as well as specific and negative evidence. In fact, we describe how to enter likelihood evidence in both Netica and Hugin in Figure 3.15. In our opinion, the explanations of this sort of evidence in both the literature and available software documentation are confusing and incomplete. It is important for people using this feature in the software to understand how likelihood evidence affects the inference and also how to work out the numbers to enter.

First, there is the issue of how to interpret uncertainty about an observation. The uncertain information could be represented by adopting it as the new distribution over the variable in question. This would mean for the earthquake node somehow setting $Bel(Earthquake = T) = 0.8$. However, we certainly do not want to “clamp” this belief, since this probabilistic judgement should be integrated with any further independent information relevant to the presence of an earthquake (e.g., the evidence about Mary calling).

3.4.1 Using a virtual node

Let’s look at incorporating an uncertain observation for the simplest case, a single Boolean node X , with a uniform prior, that is $P(X=T)=P(X=F)=0.5$. We add a **virtual node** V , which takes values $\{T, F\}$,* as a child of X , as shown in Figure 3.5. The uncertainty in the observation of X is represented by the CPT; for an 80% sure observation this gives $P(V = T|X = T) = 0.8$ and $P(V = T|X = F) = 0.2^\dagger$. Now specific evidence is entered that $V = T$. We can use Bayes’ Theorem to perform the inference, as follows.

$$\begin{aligned} Bel(X = T) &= \alpha P(V = T|X = T)P(X = T) \\ &= \alpha 0.8 \times 0.5 \\ Bel(X = F) &= \alpha P(V = T|X = F)P(X = F) \\ &= \alpha 0.2 \times 0.5 \end{aligned}$$

Since $Bel(X = T) + Bel(X = F) = 1$, this gives us $\alpha = 2$, and hence $Bel(X = T) = 0.8$ and $Bel(X = F) = 0.2$, as desired.

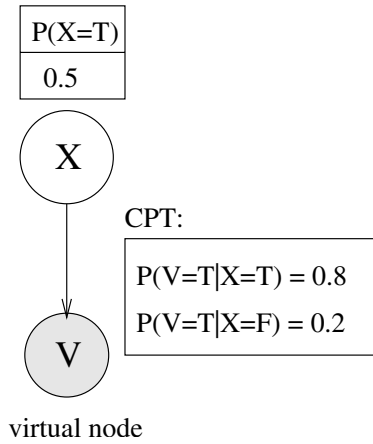
It is important to note here that due to the normalization, it is not the likelihoods for $P(V = T|X = T)$ and $P(V = T|X = F)$ that determine the new belief, but rather the **ratio**:

$$P(V = T|X = T) : P(V = T|X = F)$$

For example, we would get the same answers for $P(V = T|X) = (0.4, 0.1)$ or $P(V = T|X) = (0.5, 0.125)$.

*Note that V takes these values *irrespective* of the values of the observed node.

[†]Note that the semantics of this CPT are not well specified. For example, what is the meaning of the CPT entries for $P(V = F|X)$?

**FIGURE 3.5**

Virtual evidence handled through virtual node V for single node X , with uncertainty in CPT.

So far, we have seen that with the use of a “virtual” node we can represent uncertainty in an observation by a likelihood ratio. For example, my colleague attributing 80% credibility to there having been an earthquake means that he is four times more likely to think an earthquake occurred if that was in fact the case than he is of mistakenly thinking it occurred.

But we have considered only the situation of a node without parents, with uniform priors. If the priors are *not* uniform, then simply mapping the uncertainty into a likelihood ratio as described does not give a new belief that correctly represents the specified observational uncertainty. For example, in our earthquake example, if $P(E) = (0.02, 0.98)$, then

$$\begin{aligned}
 Bel(E = T) &= \alpha P(V = T|E = T)P(E = T) \\
 &= \alpha 0.8 \times 0.02 \\
 Bel(E = F) &= \alpha P(V = T|E = F)P(E = F) \\
 &= \alpha 0.2 \times 0.98
 \end{aligned}$$

which gives $\alpha \approx 4.72$, and $Bel(E = T) = 0.075$ and $Bel(E = F) = 0.925$ so the posterior belief in *Earthquake* is only 7.5%. However, if $P(E) = (0.9, 0.1)$, then

$$\begin{aligned}
 Bel(E = T) &= \alpha P(V = T|E = T)P(E = T) \\
 &= \alpha 0.8 \times 0.9 \\
 Bel(E = F) &= \alpha P(V = T|E = F)P(E = F) \\
 &= \alpha 0.2 \times 0.1
 \end{aligned}$$

which gives us $Bel(E = T) = 0.973$ and $Bel(E = F) = 0.027$.

It is clear that directly mapping the observational uncertainty into a likelihood ratio only results in the identical belief for the node in question (as intended in Jeffrey conditionalization) when the priors are uniform. When the priors are non-uniform, a likelihood ratio of 4:1 will increase the belief, but not necessarily to the intended degree.

Some people feel this is an advantage of this representation, supposedly indicating that the observation is an external “opinion” from someone whose priors are unknown. However, if we wish to represent unknown priors, we have no business inferring a posterior at all!

If we *really* want the uncertain evidence to shift the beliefs $Bel(X = T) = P(X = T|V) = 0.8$, then we can still use a virtual node and likelihood ratio approach, but we need to compute the ratio properly. Let’s return to our earthquake example. Recalling Definition 1.6, $P(E|V) = 0.8$ means $O(E = T|V) = 4$. Since (by odds-likelihood Bayes’ theorem 1.5)

$$O(E = T|V) = \frac{P(V = T|E = T)}{P(V = T|E = F)}O(E = T)$$

and, as given in the example,

$$O(E = T) = 0.02/0.98$$

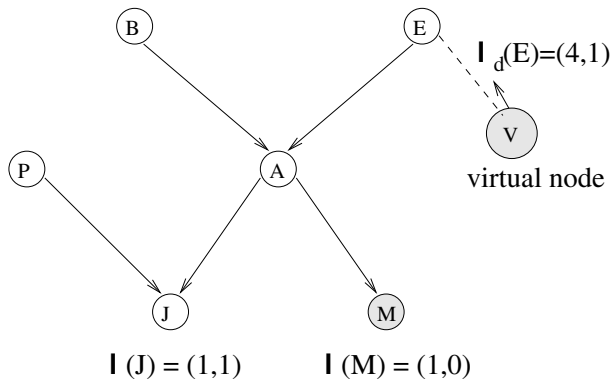
we get the likelihood ratio

$$\frac{P(V = T|E = T)}{P(V = T|E = F)} = \frac{0.8}{0.2} \times \frac{0.98}{0.02} = 196$$

This ratio is *much* higher than the one used with uniform priors, which is necessary in order to shift the belief in *Earthquake* from its very low prior of 0.02 to 0.8.

3.4.2 Virtual nodes in the message passing algorithm

When implementing a message passing inference algorithm, we don’t actually need to add the V node. Instead, the virtual node is connected by a virtual link as a child to the node it is “observing.” In the message passing algorithm, these links only carry information one way, from the virtual node to the observed variable. For our earthquake example, the virtual node V represents the virtual evidence on E . There are no parameters $\lambda(V)$, but instead it sends a $\lambda_V(E)$ message to E . The virtual node and the message it sends to E is shown in Figure 3.6.

**FIGURE 3.6**

Virtual evidence handled through virtual node V for earthquake node E , with λ message in message passing algorithm.

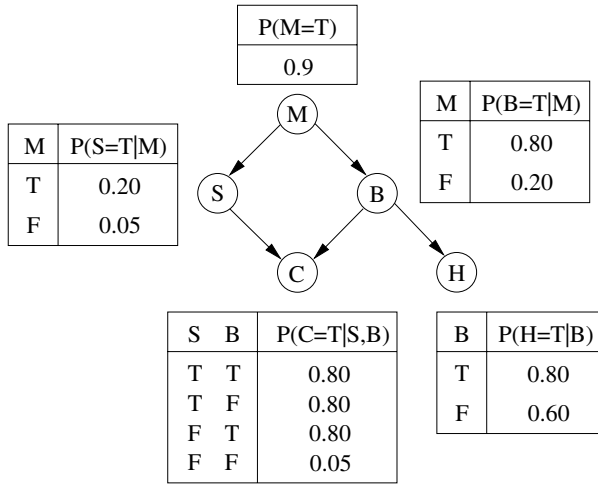
3.5 Exact inference in multiply-connected networks

In the most general case, the BN structure is a (directed acyclic) graph, rather than simply a tree. This means that at least two nodes are connected by more than one path in the underlying undirected graph. Such a network is **multiply-connected** and occurs when some variable can influence another through more than one causal mechanism.

The metastatic cancer network shown in Figure 3.7 is an example. The two causes of C (S and B) share a common parent, M . For this BN, there is an undirected loop around nodes M , B , C and S . In such networks, the message passing algorithm for polytrees presented in the previous section does not work. Intuitively, the reason is that with more than one path between two nodes, the same piece of evidence about one will reach the other through two paths and be counted twice. There are many ways of dealing with this problem in an exact manner; we shall present the most popular of these.

3.5.1 Clustering methods

Clustering inference algorithms transform the BN into a probabilistically equivalent polytree by merging nodes, removing the multiple paths between the two nodes along which evidence may travel. In the cancer example, this transformation can be done by creating a new node, say Z , that combines nodes B and S , as shown in Figure 3.8. The new node has four possible values, $\{TT, TF, FT, FF\}$, corresponding to all possible instantiations of B and S . The CPTs for the transformed network are also shown in Figure 3.8. They are computed from the CPTs of the original graph as

**FIGURE 3.7**

Example of a multiply-connected network: metastatic cancer BN (§2.5.2).

follows.

$$\begin{aligned}
 P(Z|M) &= P(S, B|M) \text{ by definition of } Z \\
 &= P(S|M)P(B|M) \text{ since } S \text{ and } B \text{ are independent given } M
 \end{aligned}$$

Similarly,

$$\begin{aligned}
 P(H|Z) &= P(H|S, B) = P(H|B) \text{ since } H \text{ is independent of } S \text{ given } B \\
 P(C|Z) &= P(C|S, B) \text{ by definition of } Z
 \end{aligned}$$

It is always possible to transform a multiply-connected network into a polytree. In the extreme case, all the non-leaf nodes can be merged into a single compound node, as shown in Figure 3.9(a) where all the nodes D_i are merged into a single super-node. The result is a simple tree. But other transformations are also possible, as shown in Figure 3.9(b) and (c), where two different polytrees are produced by different clusterings. It is better to have smaller clusters, since the CPT size for the cluster grows exponentially in the number of nodes merged into it. However the more highly connected the original network, the larger the clusters required.

Exact clustering algorithms perform inference in two stages.

1. Transform the network into a polytree
2. Perform belief updating on that polytree

The transformation in Step 1 may be slow, since a large number of new CPT values may need to be computed. It may also require too much memory if the original network is highly connected. However, the transformation only needs to be done once, unless the structure or the parameters of the original network are changed.

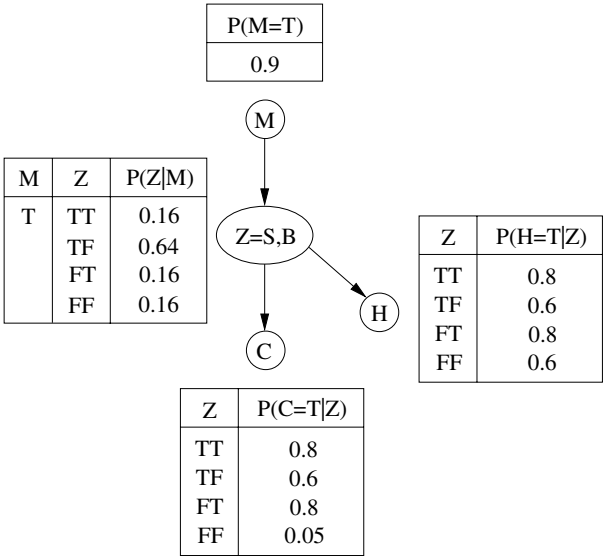


FIGURE 3.8
Result of ad hoc clustering of the metastatic cancer BN.

The belief updating in the new polytree is usually quite fast, as an efficient polytree message passing algorithm may now be applied. The caveat does need to be added, however, that since clustered nodes have increased complexity, this can slow down updating as well.

3.5.2 Junction trees

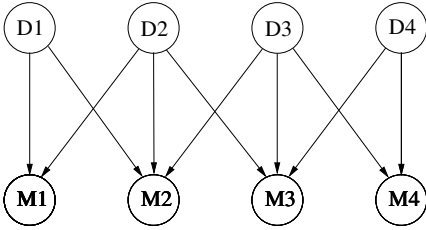
The clustering we’ve seen thus far has been ad hoc. The junction tree algorithm provides a methodical and efficient method of clustering, versions of which are implemented in the main BN software packages (see §B.4).

ALGORITHM 3.2

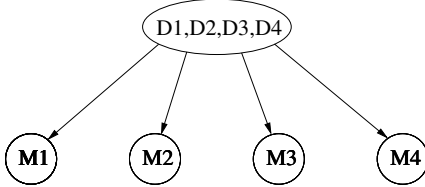
The Junction Tree Clustering Algorithm

1. **Moralize:** Connect all parents and remove arrows; this produces a so-called **moral graph**.
2. **Triangulate:** Add arcs so that every cycle of length > 3 has a chord (i.e., so there is a subcycle composed of exactly three of its nodes); this produces a **triangulated graph**.
3. **Create new structure:** Identify maximal cliques in the triangulated graph to become new compound nodes, then connect to form the so-called **junction tree**.
4. **Create separators:** Each arc on the junction tree has an attached **separator**, which consists of the intersection of adjacent nodes.

Original BN

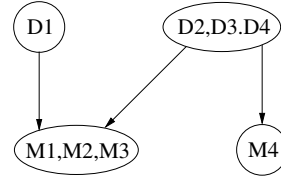


(a) Extreme Cluster – all non-leaf nodes merged

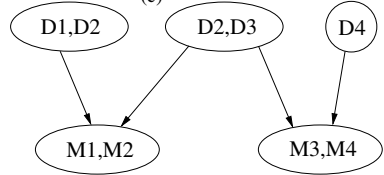


Alternative Polytrees

(b)



(c)

**FIGURE 3.9**

The original multiply-connected BN can be clustered into: (a) a tree; different polytrees (b) and (c).

5. **Compute new parameters:** *Each node and separator in the junction tree has an associated table over the configurations of its constituent variables. These are all a table of ones to start with.*

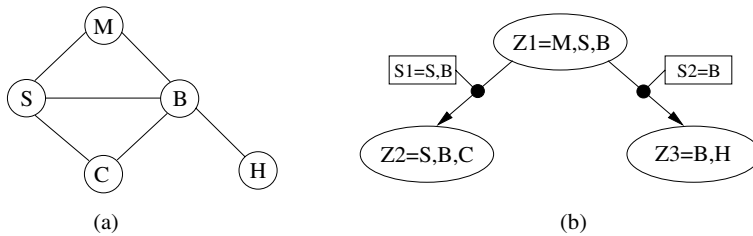
For each node X in the original network,

- (a) *Choose one node Y in the junction tree that contains X and all of X 's parents,*
- (b) *Multiply $P(X|Parents(X))$ on Y 's table.*

6. **Belief updating:** *Evidence is added and propagated using a message passing algorithm.*

The moralizing step (Step 1) is so called because an arc is added if there is no direct connection between two nodes with a common child; it is “marrying” the parent nodes. For our metastatic cancer example, S and B have C as a common child; however, there is no direct connection between them, so this must be added. The resultant moral graph is shown in Figure 3.10(a).

As we have seen, the size of the CPT for a new compound node is the product of the number of states of the combined nodes, so the size of the CPT increases exponentially with the size of the compound node. Different triangulations (Step 2) produce different clusters. Although the problem of finding an optimal triangulation is NP complete, there are heuristics which give good results in practice. No arcs need to be added to the moral graph for the metastatic cancer example, as it is already triangulated.

**FIGURE 3.10**

Applying the junction tree algorithm to the metastatic cancer BN gives (a) the moral graph and (b) the junction tree.

A **clique** of an undirected graph is defined as a set of nodes that are all pairwise linked; that is, for every pair of nodes in the set, there is an arc between them. A **maximal clique** is such a subgraph that cannot be increased by adding any node. In our example the maximal cliques can be read from the moral graph, giving three new compound nodes, $Z_1 = M, S, B$, $Z_2 = S, B, C$ and $Z_3 = B, H$ (Step 3). The junction tree, including the separators (Step 4), is shown in Figure 3.10(b).

Note that sometimes after the compound nodes have been constructed and connected, a junction *graph* results, instead of a junction tree. However, because of the triangulation step, it is always possible to remove links that aren't required in order to form a tree. These links aren't required because the same information can be passed along another path; see [129] for details.

The metastatic cancer network we've been using as an example is not sufficiently complex to illustrate all the steps in the junction tree algorithm. Figure 3.11 shows the transformation of a more complex network into a junction tree (from [129]).

Finally, Step 5 computes the new CPTs for the junction tree compound nodes. The result of Step 5 is that the product of all the node CPTs in the cluster tree is the product of all the CPTs in the original BN. Any cluster tree obtained using this algorithm is a representation of the same joint distribution, which is the product of all the cluster tables divided by the product of all the separator tables.

Adding evidence to the junction tree is simple. Suppose that there is evidence e about node X . If it is specific evidence that $X = x_i$, then we create an evidence vector with a 1 in the i th position, zeros in all the others. If it is a negative finding for state x_j , there is a zero in the j th position and ones in all the others. If it is virtual evidence, the vector is constructed as in §3.4. In all cases, the evidence vector is multiplied on the table of any node in the junction tree containing X .

Once the junction tree has been constructed (called “**compilation**” in most BN software, including Netica) and evidence entered, belief updating is performed using a message passing approach. The details can be found in [128]. The basic idea is that separators are used to receive and pass on messages, and the computations involve computing products of CPTs.

The cost of belief updating using this junction tree approach is primarily deter-

cost provides a measure that allows us to compare different junction trees, obtained from different triangulations.

While the size of the compound nodes in the junction tree can be prohibitive, in terms of memory as well as computational cost, it is often the case that many of the table entries are zeros. Compression techniques can be used to store the tables more efficiently, without these zero entries[‡].

3.6 Approximate inference with stochastic simulation

In general, exact inference in belief networks is computationally complex, or more precisely, NP hard [52]. In practice, for most small to medium sized networks, up to three dozen nodes or so, the current best exact algorithms — using clustering — are good enough. For larger networks, or networks that are densely connected, approximate algorithms must be used.

One approach to approximate inference for multiply-connected networks is **stochastic simulation**. Stochastic simulation uses the network to generate a large number of cases from the network distribution. The posterior probability of a target node is estimated using these cases. By the Law of Large Numbers from statistics, as more cases are generated, the estimate converges on the exact probability.

As with exact inference, there is a computational complexity issue: approximating to within an arbitrary tolerance is also NP hard [64]. However, in practice, if the evidence being conditioned upon is not too unlikely, these approximate approaches converge fairly quickly.

Numerous other approximation methods have been developed, which rely on modifying the representation, rather than on simulation. Coverage of these methods is beyond the scope of this text (see Bibliographic Notes §3.10 for pointers).

3.6.1 Logic sampling

The simplest sampling algorithm is that of logic sampling (LS) [106] (Algorithm 3.3). This generates a case by randomly selecting values for each node, weighted by the probability of that value occurring. The nodes are traversed from the root nodes down to the leaves, so at each step the weighting probability is either the prior or the CPT entry for the sampled parent values. When all the nodes have been visited, we have a “case,” i.e., an **instantiation** of all the nodes in the BN. To estimate $P(X|E)$ with a sample value $P'(X|E)$, we compute the ratio of cases where both X and E are true to the number of cases where just E is true. So after the generation of each case, these combinations are counted, as appropriate.

[‡]The Hugin software uses such a compression method.

ALGORITHM 3.3*Logic Sampling (LS) Algorithm***Aim:** to compute $P'(X|E = e)$ as an estimate of the posterior probability for node X given evidence $E = e$.**Initialization**

- For each value x_i for node X
 Create a count variable $\text{Count}(x_i, e)$
- Create a count variable $\text{Count}(e)$
- Initialize all count variables to 0

For each round of simulation

1. For all root nodes
 Randomly choose a value for it, weighting the choice by the priors.
2. Loop
 Choose values randomly for children, using the conditional probabilities given the known values of the parents.

 Until all the leaves are reached
3. Update run counts:
 If the case includes $E = e$
 $\text{Count}(e) \leftarrow \text{Count}(e) + 1$

 If this case includes both $X = x_i$ and $E = e$
 $\text{Count}(x_i, e) \leftarrow \text{Count}(x_i, e) + 1$

Current estimate for the posterior probability

$$P'(X = x_i|E = e) = \frac{\text{Count}(x_i, E = e)}{\text{Count}(E = e)}$$

Let us work through one round of simulation for the metastatic cancer example.

- The only root node for this network is node M , which has prior $P(M = T) = 0.2$. The random number generator produces a value between 0 and 1; any number > 0.2 means the value F is selected. Suppose that is the case.
- Next, the values for children S and B must be chosen, using the CPT entries $P(S = T|M = F)$ and $P(B = T|M = F)$. Suppose that values $S = T$ and $B = F$ are chosen randomly.
- Finally, the values for C and H must be chosen weighted with the probabilities $P(M = |S = T, B = F)$ and $P(H|S = T, B = F)$. Suppose that values $M = F$ and $H = T$ are selected.
- Then the full “case” for this simulation round is the combination of values: $C = F, S = T, B = F, M = F$ and $H = T$.

- If we were trying to update the beliefs for a person having metastatic cancer or not (i.e., $Bel(M)$) given they had a severe headache, (i.e., evidence E that $H = T$), then this case would add one to the count variable $Count(H = T)$, one to $Count(M = F, H = T)$, but not to $Count(M = T, H = T)$.

The LS algorithm is easily generalized to more than one query node. The main problem with the algorithm is that when the evidence E is unlikely, most of the cases have to be discarded, as they don't contribute to the run counts.

3.6.2 Likelihood weighting

A modification to the LS algorithm called likelihood weighting (LW) [87, 248] (Algorithm 3.4) overcomes the problem with unlikely evidence, always employing the sampled value for each evidence node. However, the same straightforward counting would result in posteriors that did not reflect the actual BN model. So, instead of adding "1" to the run count, the CPTs for the evidence node (or nodes) are used to determine how likely that evidence combination is, and that fractional likelihood is the number added to the run count.

ALGORITHM 3.4

Likelihood Weighting (LW) Algorithm

Aim: to compute $P'(X|E = e)$, an approximation to the posterior probability for node X given evidence $\mathbf{E} = e$, consisting of $\{E_1 = e_1, \dots, E_n = e_n\}$.

Initialization

- For each value x_i for node X
Create a count variable $Count(x_i, e)$
- Create a count variable $Count(e)$
- Initialize all count variables to 0

For each round of simulation

1. For all root nodes

If a root is an evidence node, E_j
choose the evidence value, e_j
 $likelihood(E_j = e_j) \leftarrow P(E_j = e_j)$

Else

Choose a value for the root node, weighting the choice by the priors.

2. Loop

If a child is an evidence node, E_j
choose the evidence value, e_j
 $likelihood(E_j = e_j) = P(E_j = e_j | \text{chosen parent values})$

Else

Choose values randomly for children, using the conditional probabilities given the known values of the parents.

Until all the leaves are reached

3. Update run counts:

If the case includes $E = e$

$$\text{Count}(e) \leftarrow \text{Count}(e) + \prod_j \text{likelihood}(E_j = e_j)$$

If this case includes both $X = x_i$ and $E = e$

$$\text{Count}(x_i, e) \leftarrow \text{Count}(x_i, e) + \prod_j \text{likelihood}(E_j = e_j)$$

Current estimate for the posterior probability

$$P'(X = x_i | E = e) = \frac{\text{Count}(x_i, E = e)}{\text{Count}(E = e)}$$

Let's look at how this works for another metastatic cancer example, where the evidence is $B = T$, and we are interested in probability of the patient going into a coma. So, we want to compute an estimate for $P(C = T | B = T)$.

- Choose a value for M with prior $P(M) = 0.2$. Assume we choose $M = F$.
- Next we choose a value for S from distribution $P(S | M = F) = 0.20$. Assume $S = T$ is chosen.
- Look at B . This is an evidence node that has been set to T and $P(B = T | M = F) = 0.05$. So this run counts as 0.05 of a complete run.
- Choose a value for C randomly with $P(C | S, B) = 0.80$. Assume $C = T$.
- So, we have completed a run with likelihood 0.05 that reports $C = T$ given $B = T$. Hence, both $\text{Count}(C = T, B = T)$ and $\text{Count}(B = T)$ are incremented.

3.6.3 Markov Chain Monte Carlo (MCMC)

Both the logic sampling and likelihood weighting sampling algorithms generate each sample individually, starting from scratch. MCMC on the other hand generates a sample by making a random change to the previous sample. It does this by randomly sampling a value for one of the non-evidence nodes X_i , conditioned on the current value of the nodes in its Markov blanket, which consists of the parents, children and children's parents (see §2.2.2).

The technical details of why MCMC returns consistent estimates for the posterior probabilities are beyond the scope of this text (see [238] for details). Note that different uses of MDMC are presented elsewhere in this text, namely Gibbs sampling (for parameter learning) in §7.3.2.1 and Metropolis search (for structure learning) in §8.6.2.

3.6.4 Using virtual evidence

There are two straightforward alternatives for using virtual evidence with sampling inference algorithms[§].

[§]Thanks to Bob Welch and Kevin Murphy for these suggestions.

1. Use a virtual node: add an explicit virtual node V to the network, as described in §3.4, and run the algorithm with evidence $V=T$.
2. In the likelihood weighting algorithm, we already weight each sample by the likelihood. We can set $likelihood(E_j)$ to the normalized likelihood ratio.

3.6.5 Assessing approximate inference algorithms

In order to assess the performance of a particular approximate inference algorithm, and to compare algorithms, we need to have a measure for the quality of the solution at any particular time. One possible measure is the **Kullback-Leibler divergence** between a true distribution P and the estimated distribution P' of a node with states i , given by[¶]:

Definition 3.1 Kullback-Leibler divergence

$$KL(P, P') = \sum_i P(i) \log \frac{P(i)}{P'(i)}$$

Note that when P and P' are the same, the KL divergence is zero (which is proven in §10.8). When $P(i)$ is zero, the convention is to take the summand to have a zero value. And, KL divergence is undefined when $P'(i) = 0$; standardly, it is taken as infinity (unless also $P(i) = 0$, in which case the summand is 0).

Alternatively, we can put this measure in terms of the updated belief for query node X .

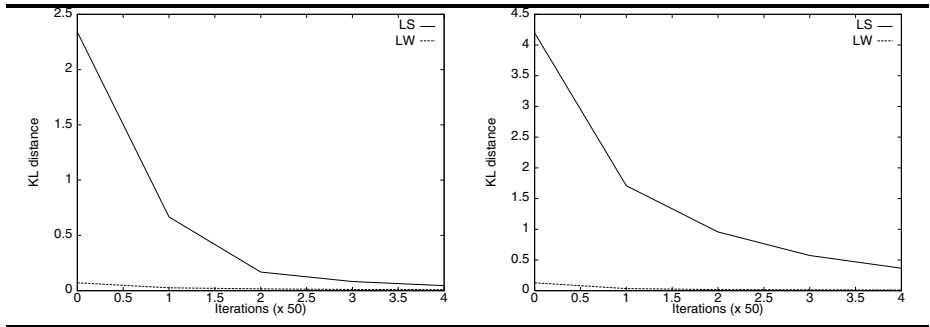
$$KL(Bel(X), Bel'(X)) = \sum_{x_i} Bel(X = x_i) \log \frac{Bel(X = x_i)}{Bel'(X = x_i)} \quad (3.7)$$

where $Bel(X)$ is computed by an exact algorithm and $Bel'(X)$ by the approximate algorithm. Of course, this measure can only be applied when the network is such that the exact posterior *can* be computed.

When there is more than one query node, we should use the marginal KL divergence over all the query nodes. For example, if X and Y are query nodes, and Z the evidence, we should use $KL(P(X, Y|Z), P'(X, Y|Z))$. Often the average or the sum of the KL divergences for the individual query nodes are used to estimate the error measure, which is not exact. Problem 3.11 involves plotting the KL divergence to compare the performance of approximate inference algorithms.

An example of this use of KL divergence is shown in Figure 3.12. These graphs show the results of an algorithm comparison experiment [205]. The test network contained 99 nodes and 131 arcs, and the LS and LW algorithms were compared for two cases:

[¶] Although “Kullback-Leibler distance” is the commonly employed word, since the KL measure is asymmetric — measuring the difference from the point of view of one or the other of the distributions — it is no true distance.

**FIGURE 3.12**

Comparison of the logic sampling and likelihood-weighting approximate inference algorithms.

- Experiment 1: evidence added for 1 root node, while query nodes were all 35 leaf nodes.
- Experiment 2: evidence added for 1 leaf node, while query nodes were all 29 root nodes (29).

As well as confirming the faster convergence of LW compared to LS, these and other results show that stochastic simulation methods perform better when evidence is nearer to root nodes [205]. In many real domains when the task is one of diagnosis, evidence tends to be near leaves, resulting in poorer performance of the stochastic simulation algorithms.

3.7 Other computations

In addition to the standard BN inference described in this chapter to data — computing the posterior probability for one or more query nodes — other computations are also of interest and provided by some BN software.

3.7.1 Belief revision

It is sometimes the case that rather than updating beliefs given evidence, we are more interested in the most probable values for a set of query nodes, given the evidence. This is sometimes called **belief revision** [217, Chapter 5]. The general case of finding a most probable instantiation of a set of n variables is called **MAP** (maximum a posteriori probability). MAP involves finding the assignment for the n variables that maximizes $P(X_1 = x_1, \dots, X_n = x_n | \mathbf{E})$. Finding MAPs was first shown to be

NP hard [254], and more recently NP complete [213]; approximating MAPs is also NP hard [1].

A special case of MAP is finding an instantiation of *all* the non-evidence nodes, also known as computing a **most probable explanation (MPE)**. The “explanation” of the evidence is a complete assignment of all the non-evidence nodes, $\{Y_1 = y_1, \dots, Y_n = y_n\}$, and computing the MPE means finding the assignment that maximizes $P(Y_1 = y_1, \dots, Y_n = y_n | \mathbf{E})$. MPE can be calculated efficiently with a similar method to probability updating (see [128] for details). Most BN software packages have a feature for calculating MPE but not MAP.

3.7.2 Probability of evidence

When performing belief updating, it is usually the case that the probability of the evidence, $P(\mathbf{E})$, is available as a by-product of the inference procedure. For example, in polytree updating, the normalizing constant α is just $1/P(\mathbf{E})$. Clearly, there is a problem should $P(\mathbf{E})$ be zero, indicating that this combination of values is **impossible** in the domain. If that impossible combination of values is entered as evidence, the inference algorithm must detect and flag it.

The BN user must decide between the following alternatives.

1. It is indeed the case that the evidence is impossible in their domain, and therefore the data are incorrect, due to errors in either gathering or entering the data.
2. The evidence should not be impossible, and the BN incorrectly represents the domain.

This notion of possible incoherence in data has been extended from impossible evidence to unlikely combinations of evidence. A **conflict measure** has been proposed to detect possible incoherence in evidence [130, 145]. The basic idea is that correct findings from a coherent case covered by the model support each other and therefore would be expected to be positively correlated. Suppose we have a set of evidence $\mathbf{E} = \{E_1 = e_1, \dots, E_m = e_m\}$. A conflict measure on \mathbf{E} is

$$C(\mathbf{E}) = \log \frac{P(E_1 = e_1), \dots, P(E_m = e_m)}{P(\mathbf{E})}$$

$C(\mathbf{E})$ being positive indicates that the evidence may be conflicting. The higher the conflict measure, the greater the discrepancy between the BN model and the evidence. This discrepancy may be due to errors in the data or it just may be a rare case. If the conflict is due to flawed data, it is possible to trace the conflicts.

3.8 Causal inference

There is no consensus in the community of Bayesian network researchers about the proper understanding of the relation between causality and Bayesian networks. The majority opinion is that there is nothing special about a **causal interpretation**, that is, one which asserts that corresponding to each (non-redundant) direct arc in the network not only is there a probabilistic dependency but also a causal dependency. As we saw in Chapter 2, after all, by reordering the variables and applying the network construction algorithm we can get the arcs turned around! Yet, clearly, *both* networks cannot be causal.

We take the minority point of view, however (one, incidentally, shared by Pearl [218] and Neapolitan [199]), that causal structure is what *underlies* all useful Bayesian networks. Certainly not all Bayesian networks are causal, but if they represent a real-world probability distribution, then some causal model is their source.

Regardless of how that debate falls out, however, it is important to consider how to do inferences with Bayesian networks that *are* causal. If we have a causal model, then we can perform inferences which are not available with a non-causal BN. This ability is important, for there is a large range of potential applications for particularly causal inferences, such as process control, manufacturing and decision support for medical intervention. For example, we may need to reason about what will happen to the quality of a manufactured product if we adopt a cheaper supplier for one of its parts. Non-causal Bayesian networks, and causal Bayesian networks using ordinary propagation, are currently used to answer just such questions; but this practice is wrong. Although the Bayesian network tools do not explicitly support causal reasoning, we will nevertheless now explain how to do it properly.

Consider again Pearl's earthquake network of Figure 2.6. That network is intended to represent a causal structure: each link makes a specific causal claim. Since it is a Bayesian network (causal or not), if we observe that *JohnCalls* is true, then this will raise the probability of *MaryCalls* being true, as we know. However, if we *intervene*, somehow forcing John to call, this probability raising inference will no longer be valid. Why? Because the reason an observation raises the probability of Mary calling is that there is a common cause for both, the Alarm; so one provides evidence for the other. However, under intervention we have effectively cut off the connection between the Alarm and John's calling. The belief propagation (message passing) from *JohnCalls* to Alarm and then down to *MaryCalls* is all wrong under **causal intervention**.

Judea Pearl, in his recent book *Causality* [218], suggests that we understand the "effectively cut off" above quite literally, and model causal intervention in a variable X simply by (temporarily) cutting all arcs from $Parents(X)$ to X . If you do that with the earthquake example (see Figure 3.13(a)), then, of course, you will find that forcing John to call will tell us nothing about earthquakes, burglaries, the Alarm or Mary — which is quite correct. This is the simplest way to model causal interventions and often will do the job.

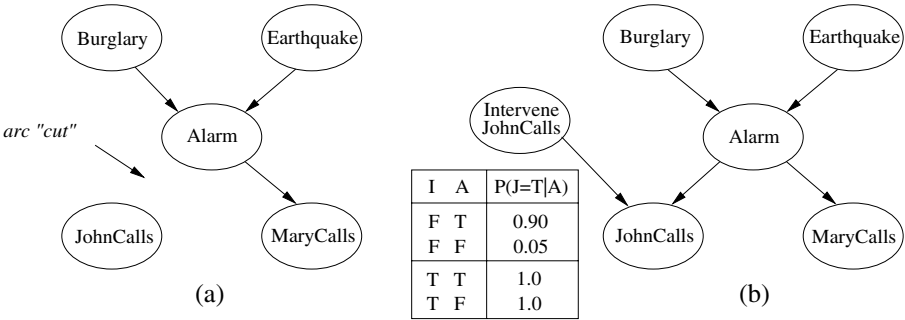


FIGURE 3.13
Modeling causal interventions: (a) Pearl’s cut method; (b) augmented model (CPT unchanged when no intervention).

There is, however, a more general approach to causal inference. Suppose that the causal intervention itself is only probabilistic. In the John calling case, we can imagine doing something which will simply *guarantee* John’s cooperation, like pulling out a gun. But suppose that we have no such guarantees. Say, we are considering a middle-aged patient at genetic risk of cardiovascular disease. We would like to model life and health outcomes assuming we persuade the patient to give up *Smoking*. If we simply cut the connection between *Smoking* and its parents (assuming the *Smoking* is caused!), then we are assuming our act of persuasion has a probability one of success. We might more realistically wish to model the effectiveness of persuasion with a different probability. We can do that by making an **augmented model**, by adding a new parent of *Smoking*, say *Intervene-on-Smoking*. We can then instrument the CPT for *Smoking* to include whatever probabilistic impact the (attempted) intervention has. Of course, if the interventions are fully effective (i.e., with probability one), this can be put into the CPT, and the result will be equivalent to Pearl’s cut method (see Figure 3.13(b)). But with the full CPT available, any kind of intervention — including one which interacts with other parents — may be represented. Subsequent propagation with the new intervention node “observed” (but not its intervened-upon effect) will provide correct causal inference, given that we have a causal model of the process in question to begin with.

Adding intervention variables will unfortunately alter the original probability distribution over the unaugmented set of variables. That distribution can be recovered by setting the priors over the new variables to zero. Current Bayesian network tools, lacking support for explicit causal modeling, will not then allow those variables to be instantiated, because of the idea that this constitutes an impossible observation; an idea that is here misplaced. As a practical matter, using current tools, one could maintain two Bayesian networks, one with and one without intervention variables, in order to simplify moving between reasoning with and without interventions^{||}.

^{||}This point arose in conversation with Richard Neapolitan.

3.9 Summary

Reasoning with Bayesian networks is done by updating beliefs — that is, computing the posterior probability distributions — given new information, called evidence. This is called probabilistic inference. While both exact and approximate inference is theoretically computationally complex, a range of exact and approximate inference algorithms have been developed that work well in practice. The basic idea is that new evidence has to be propagated to other parts of the network; for simple tree structures an efficient message passing algorithm based on local computations is used. When the network structure is more complex, specifically when there are multiple paths between nodes, additional computation is required. The best exact method for such multiply-connected networks is the junction tree algorithm, which transforms the network into a tree structure before performing propagation. When the network gets too large, or is highly connected, even the junction tree approach becomes computationally infeasible, in which case the main approaches to performing approximate inference are based on stochastic simulation. In addition to the standard belief updating, other computations of interest include the most probable explanation and the probability of the evidence. Finally, Bayesian networks can be augmented for causal modeling, that is for reasoning about the effect of causal interventions.

3.10 Bibliographic notes

Pearl [215] developed the message passing method for inference in simple trees. Kim [144] extended it to polytrees. The polytree message passing algorithm given here follows that of Pearl [217], using some of Russell's notation [237].

Two versions of junction tree clustering were developed in the late 1980s. One version by Shafer and Shenoy [252] (described in [128]) suggested an elimination method resulting in a message passing scheme for their so-called join-tree structure, a term taken from the relational data base literature. The other method was initially developed by Lauritzen and Spiegelhalter [169] as a two stage method based on the running intersection property. This was soon refined to a message passing scheme in a junction tree [124, 127, 125] described in this chapter**.

The junction tree cost given here is an estimate, produced by Kanazawa [139, 70], of the complexity of the junction tree method.

Another approach to exact inference is cutset conditioning [216, 113], where the network is transformed into multiple, simpler polytrees, rather than the single, more

**Thanks to Finn Jensen for providing a summary chronology of junction tree clustering.

complex polytree produced by clustering. This approach has not developed as a serious contender to the clustering methods implemented in BN software.

Another exact inference algorithm implemented in some current BN software is variable elimination for updating the belief of a single query node (and a variant, bucket elimination), based on product and marginalization operators; a clear exposition is given in [238]. Note that some software (e.g., JavaBayes) also refers to “bucket tree elimination,” a somewhat confusing name for what is essentially a junction tree approach.

The logic sampling method was developed by Henrion [106], while likelihood weighting was developed in [87, 248]. Other approximation methods based on stochastic sampling not covered in this book include Gibbs sampling [128], self-importance sampling and heuristic-importance sampling [248], adaptive importance sampling [43], and backward sampling [88].

There have been a number of other approximate inference methods proposed. These include state space abstraction [297], localized partial evaluation [76], weak arc removal [148] and using a mutual information measure to guide approximate evaluation [133]. It has also been shown that applying Pearl’s polytree algorithm to general networks, as suggested by Pearl [217], — so-called **loopy propagation** — can be a both fast and accurate approximate method [197]. To our knowledge, none of these methods have been implemented in widely available BN software. Hugin implements an approximation scheme that involves setting very small probabilities in the junction tree tables to zero, which in turns allows more compression to take place.

Cooper [52] showed that the general problem of inference in belief networks is NP hard, while Dagum and Luby [64] showed the problem of approximating the new beliefs to within an arbitrary tolerance is also NP hard.

3.11 Problems



Message passing


Problem 1



Consider (again — see Problem 2.8, Chapter 2) the belief network for another version of the medical diagnosis example, where B =*Bronchitis*, S =*Smoker*, C =*Cough*, X =*Positive X-ray* and L =*Lung cancer* and all nodes are Booleans. Suppose that the prior for a patient being a smoker is 0.25, and the prior for the patient having bronchitis (during winter in Melbourne!) is 0.05.

A Quick Guide to Using Hugin



Installation: Web Site www.hugin.com. Download Hugin Lite, which is available for MS Windows (95 / 98 / NT4 / 2000 / XP), Solaris Sparc, Solaris X86 and Linux. This gives you `HuginLite63.exe`, a self-extracting zip archive. Double-clicking will start the extraction process.





Network Files: BNs are stored in `.net` files, with icon . Hugin comes with a `samples` folder of example networks. To open an existing network, select , or select `File`→`Open` menu option, or double-click on the file.


Compilation: Once a Hugin BN has been opened, before you can see the initial beliefs or add evidence, you must first compile it (which they call “switch to run mode”): click on , or select `Network`→`Run`(in edit mode), or `Recompile` (in run mode) menu option.

This causes another window to appear on the left side of the display (called the Node Pane List), showing the network name, and all the node names. You can display/hide the states and beliefs in several ways. You can select a particular node by clicking on the ‘+’ by the node name, or all nodes with `View`→`Expand Node List`, or using icon . Unselecting is done similarly with ‘-’, or `View`→`Collapse Node List`, or using icon .

Selecting a node means all its states will be displayed, together with a bar and numbers showing the beliefs. Note that Hugin beliefs are given as percentages out of 100, not as direct probabilities (i.e., not numbers between 0 and 1).

Editing/Creating a BN: You can only change a BN when you are in “edit” mode, which you can enter by selecting the edit mode icon , or selecting `Network`→`Edit`. Double-clicking on a node will bring up a window showing node features, or use icon .

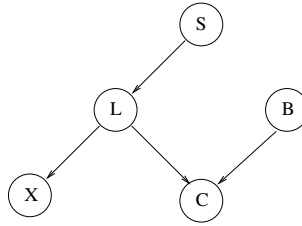
- Add a node by selecting either  (for discrete node) or  (for continuous node), `Edit`→`Discrete Chance Tool` or `Edit`→`Continuous Chance Tool`. In each case, you then “drag-and-drop” with the mouse.
- Add an arc by selecting either , or `Edit`→`Link Tool`, then left-click first on the parent node, then the child node.
- Click on the  icon to split the window horizontally between a Tables Pane (above), showing the CPT of the currently selected node, and the network structure (below).

Saving a BN: Select , or the `File`→`Save` menu option. Note that the Hugin Lite demonstration version limits you to networks with up to 50 nodes; for larger networks, you need to buy a license.

Junction trees: To change the triangulation method select `Network`→`Network Properties`→`Compilation`, then turn on “Specify Triangulation Method.” To view, select the `Show Junction Tree` option.

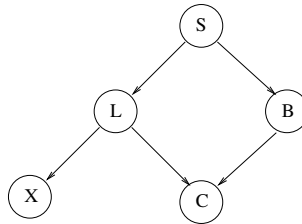
FIGURE 3.14

A quick guide to using Hugin.



Suppose that evidence is obtained that a patient has a positive X-ray result and a polytree message passing algorithm was to be used to perform belief updating.

1. Write down the π and λ values for the following nodes: S , B , C , X .
2. Show the 3 stages of data propagation by the message passing algorithm in terms of the π and λ messages.
3. Suppose that a doctor considered that smoking was a contributing factor towards getting bronchitis as well as cancer. The new network structure reflecting this model is as follows.



Why is the polytree message passing algorithm unsuitable for performing belief updating on this network?

Virtual / Likelihood evidence

Problem 2

A description of how to enter so-called likelihood evidence in both Netica and Hugin software is given in Figure 3.15. In §3.4 we only looked at an example where we added virtual evidence for a root node; however it can be added for any node in the network. Consider the cancer example from Chapter 2 (supplied as Netica file `cancer.dne`). Suppose that the radiologist who has taken and analyzed the X-ray in this cancer example is uncertain. He thinks that the X-ray looks positive, but is only 80% sure.

- Add this evidence as a likelihood ratio of 4:1.
- Work out the likelihood ratio that needs to be used to produce a new belief of $Bel(X\text{-ray}) = 0.8$
- Add this likelihood ratio as evidence to confirm your calculations.
- Add an explicit virtual node V to represent the uncertainty in the observation. Confirm that adding specific evidence for V gives the same results as adding the likelihood evidence.

Adding virtual evidence

Suppose you want to add uncertain evidence for node X with values $\{x_1, \dots, x_n\}$ and that you have worked out that the likelihood ratio vector is $(r_1 : r_2 : \dots : r_n)$ (as described in §3.4).

Netica


Right click on node and select the `likelihood` option. This will bring up a series of windows, the first of which asks you to provide a probability (default is 1) for $P(V|X = x_1)$.

Here you should enter the probability r_1 . Note that this *is* a probability, unlike everywhere else in Netica, where probabilities are given as values out of 100. You are asked for a series of these probabilities, one for each possible value of X .

If you try to re-enter likelihood evidence for a node, you will be asked if you want to discard previous evidence. If you do not, both the old and new evidence will be incorporated, equivalent to using multiple virtual nodes for each piece of evidence.

Hugin

To add evidence in Hugin, you have the following options.

- With a node selected, select `Network`→`Enter Likelihood`.
- Right click on belief bar/belief/state entry, and then choose the `Enter Likelihood` option.
- Click on the icon .

This brings up a window, showing each state with a horizontal bar whose length represents the evidence with a value from 0 to 1. The default for all states is 1.

- To set specific evidence, set a single value x_i to 1.0, and all the others to 0.
- To set negative evidence, set a single value x_j to 0, and leave all the others 1.
- To set likelihood evidence, set each value x_k corresponding to the r_k ratio that you have previously determined.

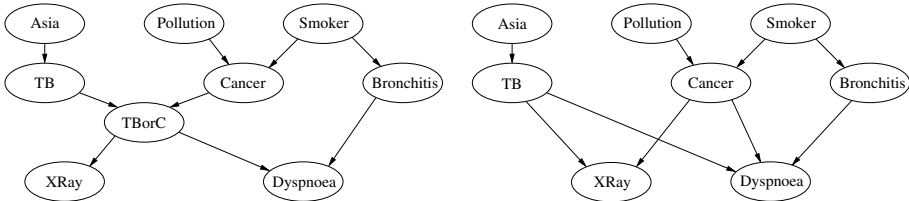
FIGURE 3.15

Adding virtual evidence in Netica and Hugin.

Clustering

Problem 3

Consider the BNs for the “Asia” problem described in §2.5.3.



1. Use the Jensen junction tree algorithm (Algorithm 3.2) to construct a junction tree from both these networks, drawing (by hand) the resultant junction trees.
2. Load these networks (`asia1.dne` and `asia2.dne` in the on-line material) into Netica.
 - (a) Compile the networks using standard Netica compilation.
 - (b) Inspect the junction tree by selecting `Report→Junction tree` menu option, and note the elimination ordering (used in Netica’s junction tree algorithm) by selecting `Report→Elimination Ordering` menu option. How do the junction trees produced by Netica compare to the ones you computed using Algorithm 3.2? What are the junction tree costs of each?
 - (c) Re-compile the networks using the `Network→Compile Optimizing` menu option. Inspect again the junction trees and the elimination orderings. How much do they differ?
3. Now load the Hugin versions of these networks (`asia.net` and `asia2.net` in the on-line material), and compile them with varying settings of the triangulation heuristic. (See **A Quick Guide to Using Hugin Expert** in Figure 3.14.)
 - (a) Clique Size
 - (b) Clique Weight
 - (c) Fill-in Size
 - (d) Fill-in Weight
 - (e) Optimal Triangulation

How do the resultant junction trees differ in structure and corresponding junction tree cost

- (a) From each other?
- (b) From the those you obtained executing the algorithm by hand?
- (c) From those obtained from Netica’s standard and optimized compilation?

Approximate inference

Problem 4

The on-line material for this text provides a version of both the Logic Sampling algorithm (Algorithm 3.3) and the Likelihood Weighting algorithm (Algorithm 3.4).

Take an example BN (either provided with the online material, or that you've developed for the problems set in Chapter 2) and do the following.

1. Run the BN software to obtain the exact inference result.
2. Run the LS Algorithm, printing out the approximate beliefs every 10 iterations and stopping when a certain level of convergence has been achieved.
3. Do the same for the LW algorithm.
4. As we have seen, the Kullback-Leibler divergence (§3.6.5) can be used to measure the error in the beliefs obtained using an approximate inference algorithm. Compute and plot the KL error over time for both the LS and LW algorithm.
5. Investigate what effect the following changes may have on the error for the LW algorithm.
 - Vary the priors between (i) more uniform and (ii) more extreme.
 - Vary the location of the evidence (i) root, (ii) intermediate and (iii) leaf.
 - Set evidence that is more or less likely.

Problem 5

As mentioned in the Bibliographic Notes, Hugin implements an approximation scheme that involves setting very small probabilities in the junction tree tables to zero, which in turns allows more compression to take place.

To turn on the approximation scheme, select `Network→Network Properties→Compilation`, then check the approximate optimization box.

As for the previous problem, select an example BN and perform an investigation of the effect of varying the approximation threshold on the belief updating, again using the KL divergence measure.

Causal reasoning

Problem 6

Take Pearl's earthquake example. Suppose there is an intervention on *JohnCalls*.

1. Use ordinary observation and updating. What is $Bel(Burglary)$?
2. Use Pearl's cut-link method. What is $Bel(Burglary)$? this case?
3. Use the augmented model with 0.9 effectiveness. What is $Bel(Burglary)$?

Now add an independent observation for $MaryCalls=T$. Parts 4, 5 and 6 of this problem involve repeating the parts 1, 2 and 3 for this new situation.

