4

Belief Updating in Bayesian Networks

In this chapter, we present algorithms for probability updating. An efficient updating algorithm is fundamental to the applicability of Bayesian networks. As shown in Chapter 2, access to $P(\mathcal{U}, e)$ is sufficient for the calculations. However, because the joint probability table increases exponentially with the number of variables, we look for more-efficient methods. Unfortunately, no method guarantees a tractable calculational task. However, the method presented here represents a substantial improvement, and it is among the most-efficient methods known.

We shall use the framework of potentials. A conditional probability table $P(A \mid \mathrm{pa}(A))$ is a function $\phi : \mathrm{pa}(A) \cup \{A\} \to [0:1]$, and we call it a potential. For the algebra of probability tables we shall for notational convenience use functional notation. That is, the product $P(A \mid \mathrm{pa}(A)) \cdot P(B \mid \mathrm{pa}(B))$ is considered as a product of two functions $\phi_1(A, \mathrm{pa}(A))\phi_2(B, \mathrm{pa}(B))$. The reader is expected to be familiar with Section 1.4.

Sections 4.1–4.6 present the junction tree algorithm, a version of the variable elimination method. Section 4.7 presents an alternative method with any-space properties, recursive conditioning, and in Sections 4.8 and 4.9 we outline different approximation methods.

4.1 Introductory Examples

To repeat the fundamentals from Chapter 2 and for pinpointing the issues in belief updating for Bayesian networks, we consider in this section a simple example. Consider the Bayesian network in Figure 4.1 over the universe $\mathcal{U}$. The potentials specified for BN are $\phi_1 = P(A_1), \phi_2 = P(A_2 \mid A_1), \phi_3 = P(A_3 \mid A_1), \phi_4 = P(A_4 \mid A_2), \phi_5 = P(A_5 \mid A_2, A_3)$, and $\phi_6 = P(A_6 \mid A_3)$.

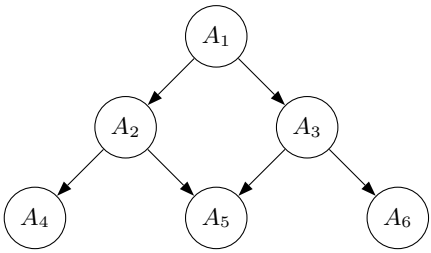


Fig. 4.1. A simple Bayesian network, BN.

4.1.1 A Single Marginal

Let us first assume that we wish to calculate $P(A_4)$. From the chain rule, we have

$$P(\mathcal{U}) = \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6 \text{ and } P(A_4) = \sum_{A_1,A_2,A_3,A_5,A_6} \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6.$$

To avoid calculating $P(U)$, we use the distributive law (Section 1.4):

$$P(A_4) = \sum_{A1} \phi_1(A_1) \sum_{A_2} \phi_2(A_2, A_1)\phi_4(A_4, A_2) \sum_{A_3} \phi_3(A_3, A_1) \sum_{A_5} \phi_5(A_5, A_2, A_3) \sum_{A_6} \phi_6(A_6, A_3).$$

First, calculate $\phi'_6(A_3) = \sum_{A_6} \phi_6(A_6, A_3)$, then multiply $\phi'_6(A_3)$ by $\phi_5(A_5, A_2, A_3)$ and calculate $\phi'_5(A_2, A_3) = \sum_{A_5} \phi_5\ (A_5, A_2, A_3)\ \phi'_6(A_3)$; $\phi'_5(A_2, A_3)$ is multiplied by $\phi_3(A_3, A_1)$, and so forth. Notice that in the calculation of $\phi'_5(A_2, A_3)$ you can apply the distributive law again; that is, you need not multiply by $\phi'_6(A_3)$ before you marginalize A_3 out. The calculation is sketched graphically in Figure 4.2.

The reason for using the distributive law is to reduce the size of the tables to handle. The full joint, $P(\mathcal{U})$, requires a space incorporating all six variables. For the process illustrated in Figure 4.2, the largest potential to handle contains three variables. In Figure 4.3, the structure is repeated, but in each bucket (drawn as an ellipse) we have indicated the variables to handle, and the variables in a mailbox (drawn as a rectangle) indicate the domain of the potential communicated.

In the preceding calculations, we performed the marginalizations in a particular order, namely A_6, A_5, A_3, A_2, A_1, and this is reflected in the structure of Figure 4.2. Because marginalization is commutative (Section 1.4), it can be done in any order. It is standard to use the term *elimination order* rather than marginalization order. If we use the reversed elimination order, we get the structure in Figure 4.4.

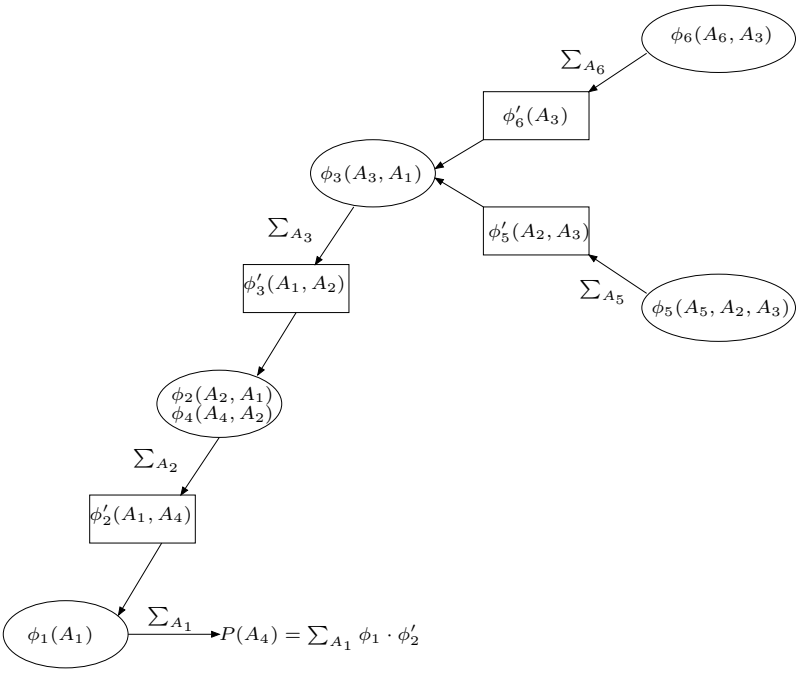


Fig. 4.2. An illustration of the process of marginalizing down to A_4. The elliptic nodes are buckets containing potentials. In a bucket, the potentials are multiplied by the incoming potentials, a variable is marginalized out, and the result is placed in a mailbox (a rectangular node) for a neighboring bucket.

Figure 4.5 illustrates the domains to handle for the last elimination order. As can be seen, the domains for the first order are smaller than the domains for the last order.

Because the size of the domains to handle is a good measure of complexity, we will address the task of finding an elimination order yielding the smallest domains to handle.

4.1.2 Different Evidence Scenarios

In the preceding calculations, we assumed that no evidence was entered into the network. By analyzing the process illustrated in Figure 4.2, we realize some simplifications. Because $\phi_5 = P(A_5 \mid A_2, A_3)$ and $\phi_6 = P(A_6 \mid A_3)$, we have that $\phi'_5 = \sum_{A_5} P(A_5 \mid A_2, A_3) = \mathbf{1}$ and $\phi'_6 = \sum_{A_6} P(A_6 \mid A_3) = \mathbf{1}$, where $\mathbf{1}$ is the unit potential. Also,

$$\phi'_3 = \sum_{A_3} \phi_3 \phi'_5 \phi'_6 = \sum_{A_3} \phi_3 \mathbf{1} \cdot \mathbf{1} = \sum_{A_3} \phi_3 = \sum_{A_3} P(A_3 \mid A_1) = \mathbf{1}.$$

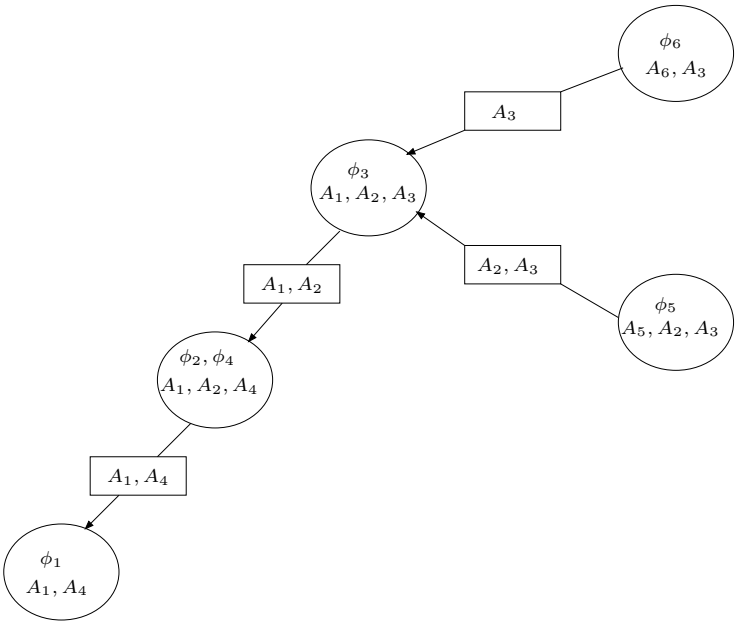


Fig. 4.3. A structure indicating the domains of the various potentials to handle.

We note that ϕ'_3 is void, and the entire process is reduced to calculating $\sum_{A_1} P(A_1) \sum_{A_2} P(A_2 \mid A_1) P(A_4 \mid A_2)$.

The nodes A_3, A_5, and A_6 are examples of so-called *barren nodes*.[1] A node A is barren if neither A nor any of A's descendants have received evidence. The conditional probability potential attached to a barren node has an impact only on descendant nodes.

If we have the evidence $A_5 = a_5$ and $A_6 = a_6$, the evidence is represented as two 0-1 findings, $\mathbf{e}_5$ and $\mathbf{e}_6$ (Section 2.3.3). The formula is

$$P(A_4, e) = \sum_{A_1, A_2, A_3, A_5, A_6} \phi_1 \phi_2 \phi_3 \phi_4 \phi_5 \phi_6 \mathbf{e}_5 \mathbf{e}_6,$$

and we have (Section 2.3.3)

$$P(A_4 \mid e) = \frac{P(A_4, e)}{\sum_{A_4} P(A_4, e)}.$$

To calculate $P(A_4, e)$, the effect on the frame in Figure 4.3 is that the two evidence potentials are added in the buckets with ϕ_5 and ϕ_6 attached to them (see Figure 4.6).

[1] This term was first used in connection with influence diagrams (Section 9.4), where barren nodes have no influence on the decisions.

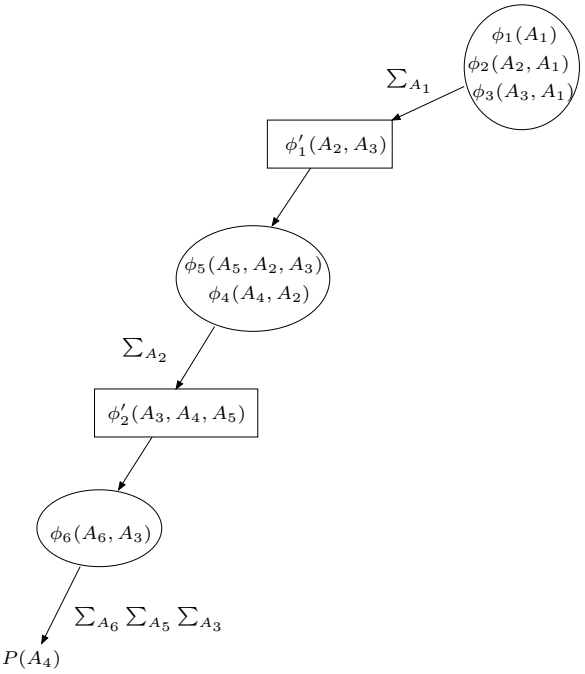


Fig. 4.4. The structure resulting from eliminating in an order that is the reverse of that from Figure 4.2.

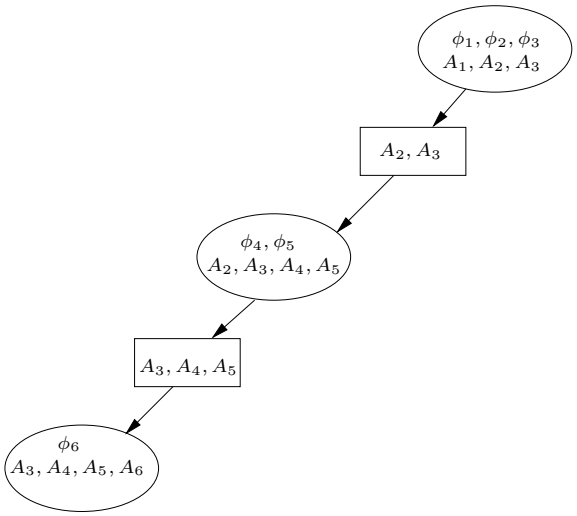


Fig. 4.5. The domains for the elimination order A_1, A_2, A_3, A_5, A_6.

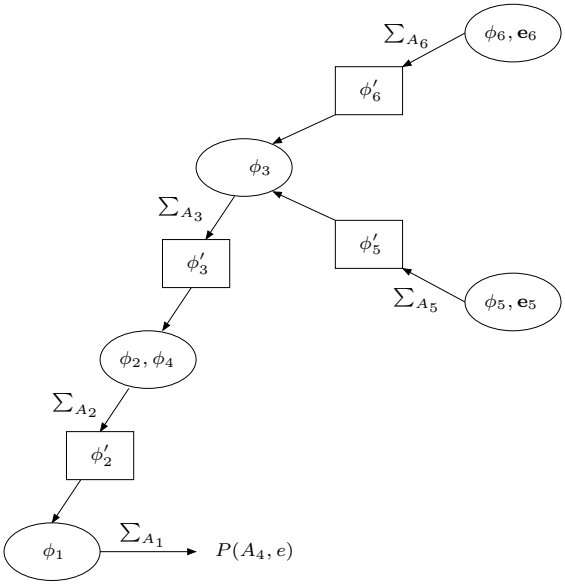


Fig. 4.6. The frame from Figure 4.2 incorporating the evidence $e_5 : A_5 = a_5$ and $e_6 : A_6 = a_6$.

The effect of e is that the variables A_5 and A_6 are instantiated in the potentials ϕ_5 and ϕ_6, and the marginalizations of A_5 and A_6 are redundant, that is, $\phi'_5 = P(A_5 = a_5 \,|\, A_2, A_3)$ and $\phi'_6 = P(A_6 = a_6 \,|\, A_3)$.

The process in Section 4.1.1 is sufficiently general to encompass all types of evidence scenarios. The task is to supplement this general process with methods taking advantage of simplifications due to the particular evidence scenario, such as identification of barren nodes.

4.1.3 All Marginals

Assume that we wish to compute $P(A_i, e)$ for all i. Without taking advantage of the special evidence scenario, we can for each node use the method from Section 4.1.1. Assume that we calculate $P(A_2, e)$ through the elimination order A_6, A_5, A_3, A_1, A_2. Then, the frame of potentials looks as in Figure 4.7.

As can be seen, the frame in Figure 4.7 is very similar to the frame in Figure 4.3. Only one arrow is reversed, and many calculations from the calculation of $P(A_4, e)$ can be reused. In this chapter, we present a systematic way of exploiting reuse in calculating all marginals. The resulting method has a complexity equivalent to two single-variable marginalizations.

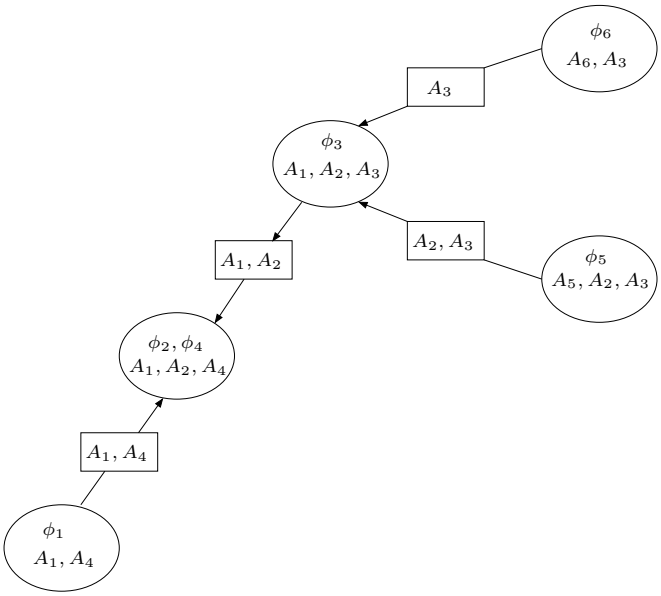


Fig. 4.7. A frame for computing $P(A_2, e)$ through the elimination order A_6, A_5, A_3, A_1, A_2.

4.2 Graph-Theoretic Representation

As illustrated in Section 4.1, belief updating for Bayesian networks consists basically in calculating sums of products. In this section, we deal systematically with this task without explicit reference to Bayesian networks. The methods presented are general and can be applied to a large variety of tasks.

4.2.1 Task and Notation

We will work with a set of real-valued potentials $\Phi = \{\phi_1, \ldots, \phi_m\}$ over finite variables from the universe $\mathcal{U} = \{A_1, \ldots, A_n\}$.

Let Ψ be any set of potentials. The product of all potentials ψ in Ψ is denoted by $\prod \psi$. We will also use the notation $\prod_{i=1}^{k} \psi_i$ for the product $\psi_1 \cdots \psi_k$, and if the boundaries are apparent from the context, we write $\prod \psi_i$.

The potential $\sum_X \phi(X, Y, \ldots, Z)$ is the sum $\phi(x_1, Y, \ldots, Z) + \cdots + \phi(x_k, Y, \ldots, Z)$, and it is defined over $(Y, \ldots, Z)$. We say that X has been *marginalized* out of $\phi(X, Y, \ldots, Z)$. If $\mathcal{V}$ is a set of variables, then $\sum_{\mathcal{V}}$ is a notation for marginalizing out all variables in $\mathcal{V}$. Because marginalization is commutative (Section 1.4), this notation is unambiguous.

Instead of sum notation, we may also use projection notation. We let $\phi^{\downarrow X}(X, Y, \ldots, Z)$ denote the potential resulting from marginalizing out $(Y, \ldots, Z)$; the potential is projected down to X. If $\mathcal{W}$ is a set of variables, then $\phi^{\downarrow \mathcal{W}}$

denotes the result of marginalizing out all variables except the members of $\mathcal{W}$.

Task: Compute $(\prod \Phi)^{\downarrow A_i}$ for all A_i.

Definition 4.1. *Let Φ be a set of potentials, and let X be a variable. Then X is* eliminated from Φ *through the following procedure:*

1. *Remove all potentials in Φ with X in their domains. Call the set of removed potentials Φ_X.*
2. *Calculate $\phi^{-X} = \sum_X \prod \Phi_X$.*
3. *Add ϕ^{-X} to Φ. Call the result Φ^{-X}; $\Phi^{-X} = \{\Phi \setminus \Phi_X, \phi^{-X}\}$.*

Note that elimination of the variable X corresponds to using the distributive law on the product. Instead of calculating the product, we keep the factors in a bucket and do not multiply before we are forced to do so.

Proposition 4.1. *The task $(\prod \Phi)^{\downarrow X}$ is solved by repeatedly eliminating the variables except for X.*

It remains to establish an elimination order.

4.2.2 Domain Graphs

To get an overview of the consequences of various elimination orders, the task is represented graphically.

Definition 4.2. *Let $\Phi = \{\phi_1, \ldots, \phi_m\}$ be potentials over $\mathcal{U} = \{A_1, \ldots, A_n\}$ with* $\mathrm{dom}(\phi_i) = \mathcal{D}_i$. *The* domain graph *for Φ is the undirected graph with the variables of $\mathcal{U}$ as nodes and with a link between each pair of variables that are members of the same $\mathcal{D}_i$.*

For the sake of exposition, we assume throughout the chapter that the graphs considered are connected.

Example 4.1. In Section 4.1.1, we dealt with a Bayesian network over the potentials $\Phi = \{\phi_1(A_1), \phi_2(A_2, A_1), \phi_4(A_4, A_2), \phi_3(A_3, A_1),\ \phi_5(A_5, A_2, A_3), \phi_6(A_6,\ A_3)\}$. The domain graph for Φ is given in Figure 4.8.

Compared to the initial Bayesian network in Figure 4.1, we see that directions on the links have been dropped and that a new link (A_2, A_3) has been inserted. It is often called a *moral link* because it connects two nodes with a common child. The domain graph for a Bayesian network is called the *moral graph*.

When we eliminate a variable X, we work with the product of all potentials with X in the domain. The domain of this product consists of X and its neighbors in the domain graph, and when X is eliminated, the resulting potential has all X's neighbors in its domain. This means that in the domain

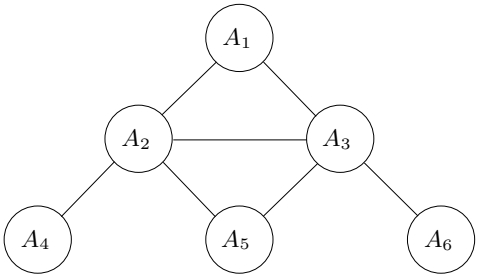


Fig. 4.8. The domain graph for $\Phi = \{\phi_1(A_1), \phi_2(A_2, A_1), \phi_3(A_3, A_1), \phi_4(A_4, A_2), \phi_5(A_5, A_2, A_3), \phi_6(A_6, A_3)\}$.

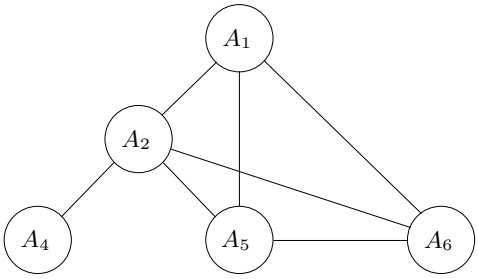


Fig. 4.9. The domain graph for Φ^{-A_3} from Figure 4.8.

graph for Φ^{-X} all neighbors of X are pairwise linked. In Figure 4.9, we show the domain graph for the example in Figure 4.8 with A_3 eliminated.

Note that the graph in Figure 4.9 has several new links. These new links are called *fill-ins*. The introduction of fill-ins highlights the fact that when eliminating A_3 you work with a potential over a domain that was not present initially. In order to avoid working with new domains, you try to avoid fill-ins. To put it another way, an elimination sequence that does not introduce fill-ins requires less space than an elimination sequence that introduces fill-ins.

In Section 4.1, we considered calculation of $P(A_4)$. In the graph-theoretic framework, it corresponds to constructing an elimination sequence ending with A_4. For the domain graph in Figure 4.8, it is possible to eliminate down to A_4 without introducing fill-ins: $A_6, A_5, A_3, A_1, A_2, A_4$. Such a sequence is called a *perfect elimination sequence*. There are several perfect elimination sequences ending with A_4, and an optimal elimination sequence will be found among them. In Figure 4.8, we see that the sequence $A_5, A_6, A_3, A_1, A_2, A_4$ as well as $A_1, A_5, A_6, A_3, A_2, A_4$ and $A_6, A_1, A_3, A_5, A_2, A_4$ are perfect elimination sequences.

Proposition 4.2. *Let $X_1, \ldots, X_k$ be a perfect elimination sequence, and let X_j be a node with a complete neighbor set.*[1] *Then, the sequence $X_j, X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_k$ is also a perfect elimination sequence.*

Proof. If you start by eliminating X_j, you do not introduce fill-ins. Consider variable X_i. When you eliminate X_i, you look at the uneliminated neighbors, and if a pair of them is not linked, you introduce a fill-in. Eliminating X_j before X_i does not give X_i new neighbors, and it will not enforce new fill-ins when X_i is eliminated. □

The complexity of using a particular elimination sequence is characterized by the set of domains for the potentials used. For the elimination order $A_6, A_5, A_3, A_1, A_2, A_4$, the set of domains is $\{\{A_6, A_3\}, \{A_2, A_3, A_5\}, \{A_1, A_2, A_3\}, \{A_1, A_2\}, \{A_2, A_4\}\}$. If a domain is a subset of another domain, then it does not require extra space and we need not consider it. For example, the set $\{A_1, A_2\}$ is removed from the preceding domain set.

Definition 4.3. *The* domain set *of an elimination sequence is the set of domains of potentials produced during the elimination in which potentials that are subsets of other potentials are removed.*

Unfortunately, it does not hold that if you eliminate without introducing fill-ins, then the domain set consists only of domains from the initial set of potentials. For the preceding perfect elimination sequence, we have that when A_3 is eliminated, you work with a potential with domain $\{A_1, A_2, A_3\}$, which is not one of the initial domains. However, there is no way to avoid this. No matter which of the three variables you eliminate first, you will produce a potential with all three variables in the domain. In general, it holds that if the set $\mathcal{V}$ of variables is a complete set in the domain graph, then any elimination sequence will contain a potential with a domain including $\mathcal{V}$.

Proposition 4.3. *All perfect elimination sequences produce the same domain set, namely the set of cliques of the domain graph; a complete set is a* clique *if it is not a subset of another complete set (a maximal complete set).*

Proof. First we show that a clique V corresponds to the domain of a potential produced during the elimination. Let X be the first variable from V to be eliminated. When X is eliminated, we produce a domain D consisting of X and all its neighbors. Because all elements of V are neighbors of X, D must contain V. Let Y be a member of D. After elimination of X, there is a link between Y and all members of V. The elimination does not produce fill-ins, so the links must have been present initially, and because V is a maximal complete set, Y must be a member of V. Hence, the cliques must be members of the domain set.

[1] A set of nodes is *complete* if all nodes are pairwise linked.

Finally we show that each member W of the domain set is a clique. Because the elimination does not produce fill-ins, W must be a complete set in the domain graph. If W is not maximal, it is a subset of a clique V, and V is a member of the domain set, so W cannot be a member. □

From Proposition 4.3, we can conclude that any perfect elimination sequence ending with the variable A is optimal with respect to calculating $P(A)$. The full task is to compute the marginals down to each variable, so the task can be solved by establishing an optimal elimination sequence for each variable.

4.3 Triangulated Graphs and Join Trees

Before continuing with the belief-updating task, we deal in detail with some purely graph-theoretic concepts. They will be used for the belief updating task in the next section.

Definition 4.4. *An undirected graph with a perfect elimination sequence is called a* triangulated graph.

Note that the term "triangulated" may be misleading. The graph (b) in Figure 4.10 is not triangulated.

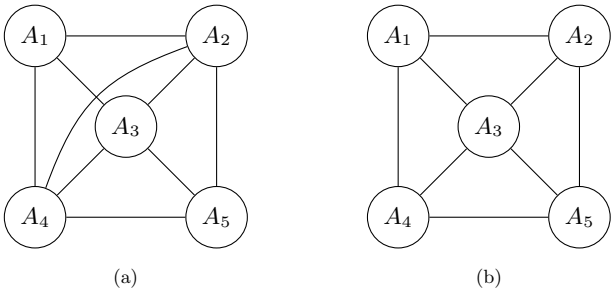


Fig. 4.10. (a) A triangulated graph; (b) a nontriangulated graph.

Notation: Let X be a node in an undirected graph. The set of neighbors of X we denote by $\mathrm{nb}(X)$, and the set of neighbors plus X we denote by $\mathrm{fa}(X)$, the family of X. If the nodes of the graph are enumerated, we use the index to write N_i rather than N_{X_i}. Nodes with a complete neighbor set are called *simplicial nodes*. A neighbor to a node X is said to be *adjacent* to X. Note that X is simplicial if and only if $\mathrm{fa}(X)$ is a clique.

Proposition 4.4. *Let G be a triangulated graph, and let X be a simplicial node. Let G' be the graph resulting from eliminating X from G (see Figure 4.11). Then G' is a triangulated graph.*

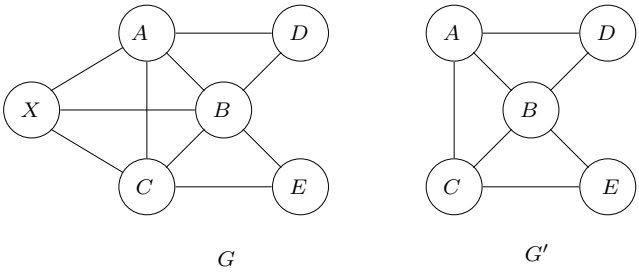


Fig. 4.11. If $\mathrm{fa}(X)$ is a complete set, you eliminate X from G by simply removing X together with its links.

Proof. Follows from Proposition 4.2. □

Note that a triangulated graph always has at least one simplicial node, namely the first one in the elimination sequence. Actually, there are at least two.

Theorem 4.1. *A triangulated graph with at least two nodes has at least two simplicial nodes.*

Proof. We prove by induction a slightly stronger statement: let G be an incomplete triangulated graph with at least three nodes. Then, it has at least two nonadjacent simplicial nodes.

Certainly, any incomplete triangulated graph with three nodes has two nonadjacent simplicial nodes (see Figure 4.12).

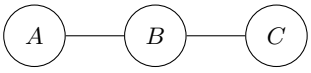


Fig. 4.12. A connected incomplete triangulated graph with three nodes.

Assume the statement to be true for all graphs with fewer than n nodes, and let G be an incomplete triangulated graph with n nodes. The first node, X, in the elimination sequence is simplicial, and we must find another one not adjacent to X. Let G' be the graph resulting from removing X from G.

The graph G' is triangulated, and any simplicial node in G' is either simplicial in G or a member of $\mathrm{nb}(X)$.

Because G is not complete, it must contain nodes that are not members of $\mathrm{nb}(X)$. If G' is complete, any of these nodes can do. If G' is not complete, we know from the induction hypothesis that it has at least two nonadjacent simplicial nodes. If both were neighbors of X, they would be adjacent. □

Corollary 4.1. *In a triangulated graph, each variable A has a perfect elimination sequence ending with A.*

Proof. Let A be any node in the triangulated graph G. Eliminate a simplicial node $X (X \neq A)$; Theorem 4.1 ensures that such a node exists. Proposition 4.4 yields that the resulting graph is triangulated, and you can repeatedly apply Theorem 4.1 until only A is left. □

From Corollary 4.1, we see that if you have established one perfect elimination sequence, then you can easily establish a perfect elimination sequence down to any variable. In other words, you can for each variable A establish an optimal sequence of marginalizations for calculating $P(A)$. We give the details in Section 4.4.

Unfortunately, it does not hold that all domain graphs are triangulated. The following theorem gives an easy way of checking whether a graph is triangulated, and if it is, it also gives a simple way of establishing an elimination sequence.

Theorem 4.2. *An undirected graph is triangulated if and only if all nodes can be eliminated by successively eliminating a simplicial node X.*

Proof. If all nodes can be eliminated by successively eliminating simplicial nodes, then we produce a perfect elimination sequence, and the graph is triangulated.

Now assume that the undirected graph is triangulated. Let us eliminate any simplicial node. Proposition 4.4 yields that the resulting graph is triangulated, and we can continue the procedure. □

To check whether a graph is triangulated, you repeatedly eliminate simplicial nodes. At some stage, you run into a situation in which you cannot eliminate more nodes. If the node set is empty, then the graph is triangulated; if not, then the graph is not triangulated.

In general, it is NP-hard to determine the set of cliques in a graph. For triangulated graphs, Proposition 4.3 and Theorem 4.2 yield an easy procedure.

Algorithm 4.1 *To determine the set of cliques in a triangulated graph, you can do as follows*

1. *Eliminate a simplicial node X;* $\mathrm{fa}(X)$ *is a clique candidate.*
2. *If* $\mathrm{fa}(X)$ *does not include all remaining nodes, go to 1.*
3. *Prune the set of clique candidates by removing sets that are subsets of other clique candidates.*

□

4.3.1 Join Trees

Definition 4.5. *Let $\mathcal{G}$ be the set of cliques from an undirected graph, and let the cliques of $\mathcal{G}$ be organized in a tree T. Then T is a* join tree *if for any pair of nodes V, W all nodes on the path between V and W contain the intersection $V \cap W$.*

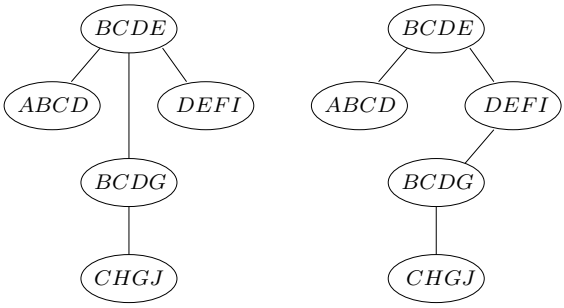


Fig. 4.13. (a) A join tree; (b) not a join tree.

Theorem 4.3. *If the cliques of an undirected graph G can be organized into a join tree, then G is triangulated.*

Proof. Let the cliques be organized in a join tree, and let V be a leaf clique with unique neighbor clique W. Any member of V that is a member of another clique must – due to the join tree condition – also be a member of W. Therefore, V must contain at least one variable X not contained in any other clique (otherwise V would be a subset of W). Then $\mathrm{fa}(X)$ must be complete, and X can be eliminated without creating fill-ins. We can repeat eliminating variables that are only members of V, and when all these have been eliminated, we have a graph G' with the same cliques as G except for V. Then, the join tree for G with the node V removed is a join tree for G', and we can continue by eliminating a variable from a leaf in G'. □

Theorem 4.4. *If the undirected graph G is triangulated, then the cliques of G can be organized into a join tree.*

The proof is a construction of a join tree from a triangulated graph. To illustrate the construction, we use the graph in Figure 4.14.

Construction: Establish an elimination sequence in the following way. Start with a simplicial node X. Then $\mathrm{fa}(X)$ is a clique. Continue eliminating nodes

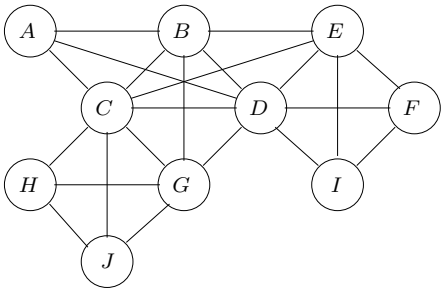


Fig. 4.14. A triangulated graph.

from fa(X) that have neighbors only in fa(X). Give fa(X) an index i according to the number of nodes eliminated, and denote the set of the remaining nodes by S_i. This set is called a *separator*. Choose a new clique in the graph G' with the eliminated nodes removed, and repeat the process with the index starting at i. Continue to do so until all cliques have been eliminated. Figure 4.15 shows the result of this process on the graph in Figure 4.14.

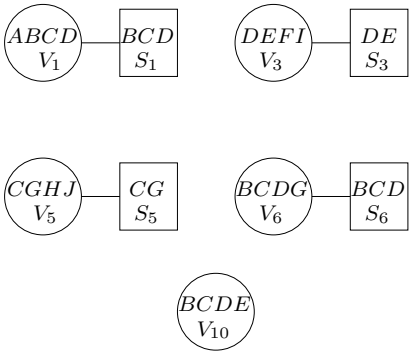


Fig. 4.15. The cliques, separators, and indices resulting from the graph in Figure 4.14. The elimination sequence used is $A, F, I, H, J, G, B, C, D, E$.

When the parts have been established as indicated in Figure 4.15, each separator S_i is connected to a clique V_j ($j > i$) such that $S_i \subset V_j$ (see Figure 4.16). This is always possible because S_i is a complete set, and when the first node from S_i is eliminated, it must be when dealing with a clique of higher index than i, and it must contain all of S_i. For convenience, we talk of the direction from V_i over S_i to V_j as upward, and we call V_j a parent of V_i.

We must prove that the structure constructed is a tree and has the join tree property. Each clique has at most one parent, so there cannot be multiple paths, and the structure is a tree.

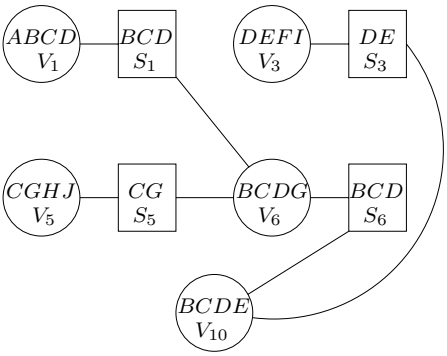


Fig. 4.16. A join tree (expanded with separators) resulting from the construction applied to the graph in Figure 4.14.

To prove the join tree condition, consider the cliques V_i and V_j $(i < j)$, and let X be a member of both. There is a unique path between V_i and V_j, and we will prove that X is a member of all cliques on that path. Because X is not eliminated when dealing with V_i, it must be a member of S_i, and from the construction, X must be a member of V_i's parent V_k. If $k = j$, we are finished; otherwise we continue the argument for the smallest of the two.

Remark: The separators are so called because any separator S divides the graph into two parts, and all paths connecting the two parts must pass through S. If the join tree is constructed from a Bayesian network, the two parts are d-separated given S.

A join tree provides the framework for constructing perfect elimination sequences. Namely, notice that the simplicial nodes are those with all uneliminated neighbors in one clique, and two nodes are neighbors if they are members of the same clique. Hence, all perfect elimination sequences can be constructed from a join tree by repeatedly eliminating simplicial nodes.

4.4 Propagation in Junction Trees

In the literature you often see the terms "join tree" and "junction tree" used interchangeably. In this book we introduce a distinction.

Definition 4.6. *Let Φ be a set of potentials with a triangulated domain graph, G. A* junction tree *for Φ is a join tree for G with the following addition: each potential ϕ in Φ is attached to a clique containing* $\mathrm{dom}(\phi)$*; each link has the appropriate separator attached; each separator contains two mailboxes, one for each direction.*

If Φ is a set of conditional probabilities for a Bayesian network BN together with evidence potentials for the evidence e, we say that the junction

tree represents BN with evidence e.

Notation: The propagation algorithm presented here deals with sets of potentials. A set of potentials is a representation of the product of the member potentials. Let Φ be a set of potentials whose domains are subsets of V, and let W be a subset of V. Then, $\Phi^{\downarrow W}$ is a set of potentials resulting from successively eliminating the variables in $V \setminus W$ as described in Definition 4.1. Because the elimination order is arbitrary, this notation seems to introduce some ambiguity with respect to the functions in the resulting set. Because we treat the sets as representations of products, and the product is independent of the elimination order, we will not deal with this apparent ambiguity.

Example 4.2. Consider the set $\psi = \{\phi_1(A),\ \phi_2(A,B),\ \phi_3(A,C),\ \phi_4(C,D),\ \phi_5(C)\}$, and let $W = \{B,C\}$. Then, $\psi^{\downarrow W} = \{\sum_A \phi_1(A)\phi_2(A,B)\phi_3(A,C),\ \sum_D \phi_4(C,\ D), \phi_5(C)\}$.

Before giving a general description of the propagation algorithm, we will go through an example.

Example 4.3. Consider the Bayesian network in Section 4.1 with potentials $\phi_1 = P(A_1), \phi_2 = P(A_2 \mid A_1), \phi_3 = P(A_3 \mid A_1), \phi_4 = P(A_4 \mid A_2), \phi_5 = P(A_5 \mid A_2, A_3), \phi_6 = P(A_6 \mid A_3)$ and with the domain graph in Figure 4.8. We know that the elimination sequence $A_6, A_5, A_3, A_1, A_2, A_4$ is perfect. The domain graph has a join tree over the cliques $V_1 = \{A_3, A_6\}$, $V_2 = \{A_2, A_3, A_5\}$, $V_4 = \{A_1, A_2, A_3\}$, $V_6 = \{A_2, A_4\}$ and the separators $S_1 = \{A_3\}$, $S_2 = \{A_2, A_3\}$, $S_4 = \{A_2\}$. The junction tree is shown in Figure 4.17.

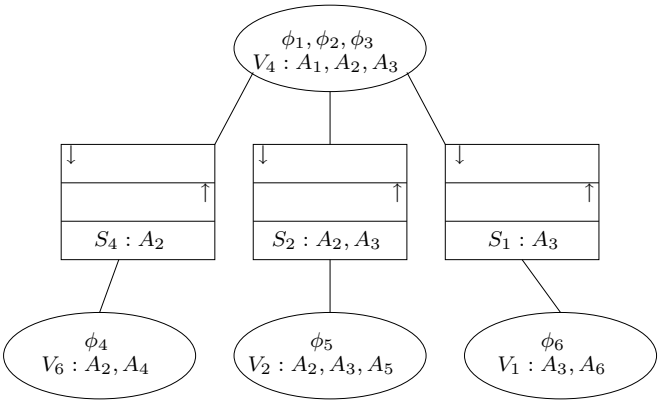


Fig. 4.17. A junction tree for the Bayesian network in Figure 4.8.

To calculate $P(A_4)$, we find a clique containing $A_4(V_6)$. It is made a temporary root, and we send messages in the direction of V_6 starting from the

leaf cliques. The message $\psi_1 = \phi_6^{\downarrow A_3} = \phi_6^{\downarrow S_1}$ is placed in the appropriate S_1 mailbox, and the message $\psi_2 = \phi_5^{\downarrow\{A_2,A_3\}} = \phi_5^{\downarrow S_2}$ is placed in the appropriate S_2 mailbox. Next, V_4 assembles the incoming messages and the potentials held form the set $\Phi_4 = \{\psi_1, \psi_2, \phi_1, \phi_2, \phi_3\}$. The variables A_1 and A_3 are eliminated from Φ_4, and the result, $\psi^4 = (\phi_1\phi_2(\phi_3\psi_2\psi_1)^{\downarrow\{A_1,A_2\}})^{\downarrow A_2} = \sum_{A_1}\phi_1\phi_2\sum_{A_3}\phi_3\psi_2\psi_1$, is placed in the appropriate mailbox (see Figure 4.18).

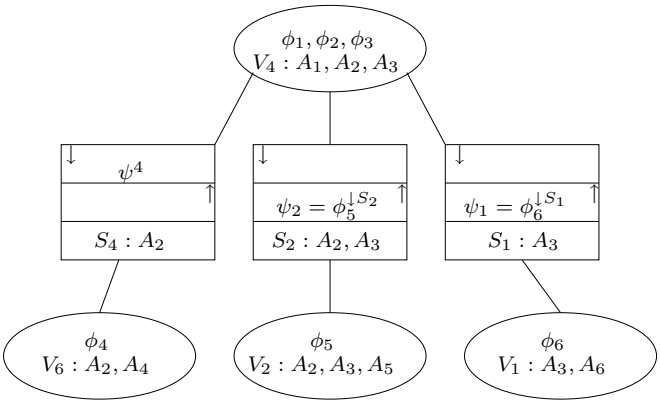


Fig. 4.18. The cliques V_1 and V_2 have sent messages to their separators, and V_4 has sent the message $\sum_{A_1}\phi_1\phi_2\sum_{A_3}\phi_3\psi_2\psi_1$ to S_4.

Now V_6 can collect its message, multiply it by ϕ_4, and marginalize A_2 out to get $P(A_4)$.

The process just described is called *collecting evidence* to V_6. To calculate the marginal for another variable X, we can collect to a clique containing X. If, for example, we wish to calculate $P(A_6)$, we can collect to V_1. We can also prepare the junction tree for the calculation of all marginals: send messages in the direction away from V_6. This process is called *distributing evidence*. First, V_6 sends the message $\psi_4 = \phi_4^{\downarrow A_2}$ to S_4, and V_4 sends a message to S_2 as well as S_1 (see Figure 4.19). When the message for S_2 is calculated, the set $\{\psi_4, \phi_1, \phi_2, \phi_3, \psi_1\}$ is assembled, and A_1 is marginalized out. Here, we multiply only the potentials that have A_1 in the domain, and the message becomes a set of potentials: $\{\psi_4, \sum_{A_1}\phi_1\phi_2\phi_3, \psi_1\}$.

When both collecting and distributing evidence have been performed, we have performed a full propagation, and to calculate a marginal $P(X)$ we find a clique V containing X. Take, for example, A_3. The clique V_1 contains A_3. The incoming message to V_1 is the message for collecting evidence to V_1, and therefore it corresponds to a perfect elimination sequence ending with the nodes A_6 and A_3. This means that the product $\phi_6\psi^1$ is the projection of the

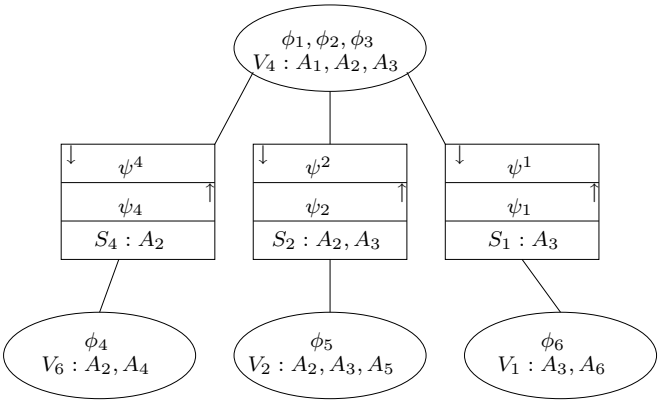


Fig. 4.19. The junction tree after a full propagation: $\psi^2 = \{\psi_4, \sum_{A_1} \phi_1\phi_2\phi_3, \psi_1\}, \psi^1 = \sum_{A_2} \psi_2\psi_4 \sum_{A_1} \phi_1\phi_2\phi_3$.

entire product down to $\{A_3, A_6\}$, and we can easily calculate $P(A_3)$ as well as $P(A_6)$.

There is a slightly easier way of calculating A_3. Consider the separator S_1. It consists of A_3 alone. For the product of the two messages of S_1, we have

$$\begin{aligned}\psi^1\psi_1 &= (\sum_{A_2} \psi_2\psi_4 \sum_{A_1} \phi_1\phi_2\phi_3)(\sum_{A_6} \phi_6) \\ &= \sum_{A_2}\sum_{A_5} \phi_5 \sum_{A_4} \phi_4 \sum_{A_1} \phi_1\phi_2\phi_3 \sum_{A_6} \phi_6 \\ &= (\phi_5\phi_4\phi_1\phi_2\phi_3\phi_6)^{\downarrow A_3} = P(A_3).\end{aligned}$$

Next, assume that you have the evidence $e = \{e_5 : A_5 = a^5, e_6 : A_6 = a^6\}$. The evidence e is represented as two 0−1 potentials $\mathbf{e}_5$ and $\mathbf{e}_6$. To calculate the probabilities $P(X, e)$, you place the two evidence potentials in appropriate cliques (V_2 and V_1) and perform a full propagation.

4.4.1 Lazy Propagation in Junction Trees

Each clique V holds a set of potentials denoted by Φ_V. Each separator has two mailboxes, one for each direction of the link. The messages stored in the mailboxes are sets of potentials. The messages are denoted by ψ_S or ψ^S, depending on the direction.

The basic operation in the lazy propagation procedure is *message passing*.

Definition 4.7. *Let V be a clique with set of potentials Φ_V, and let S be a neighboring separator. Let $S_1, \ldots, S_k$ be the other neighboring separators of V. Assume that each S_i has received a message Ψ_i for V.*

Then V can pass the message $(\Phi_V \cup \Psi_1 \cup \cdots \cup \Psi_k)^{\downarrow S}$ *to S, and we say that the direction V to S is* triggered.

The propagation method consists in repeatedly passing messages along triggered directions.

Proposition 4.5. *If you repeatedly pass messages along triggered directions in a junction tree, then you need not stop before a message has been passed in both directions over each link. In that situation, we say that the junction tree is* full.

Proof. See Exercise 4.27. □

As shown in Example 5.3, you can start off by directing all messages toward a chosen temporary root R. In other words, the junction tree is given a direction from R and outward, and the messages are passed in the opposite direction from leaves and inward (see Figure 4.20). This procedure is called CollectEvidence(R).

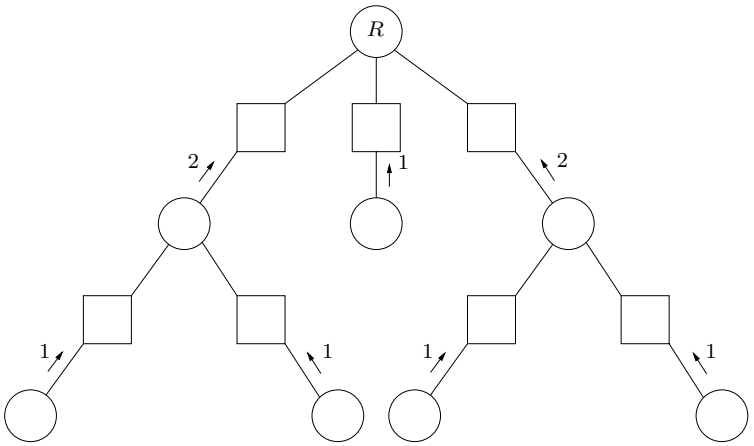


Fig. 4.20. The message passing in CollectEvidence(R).

Notice that the message passing in CollectEvidence(R) corresponds to a perfect elimination sequence ending with the nodes of R.

To fill the junction tree after a CollectEvidence(R), you need only to place messages in the opposite directions. First, R passes a message to its neighbors, they in turn pass messages further outward, and so forth out to the leaves (see Figure 4.21). This procedure is called DistributeEvidence(R). Note that messages are passed along triggered directions only if DistributeEvidence(R) is performed after CollectEvidence(R) has been performed.

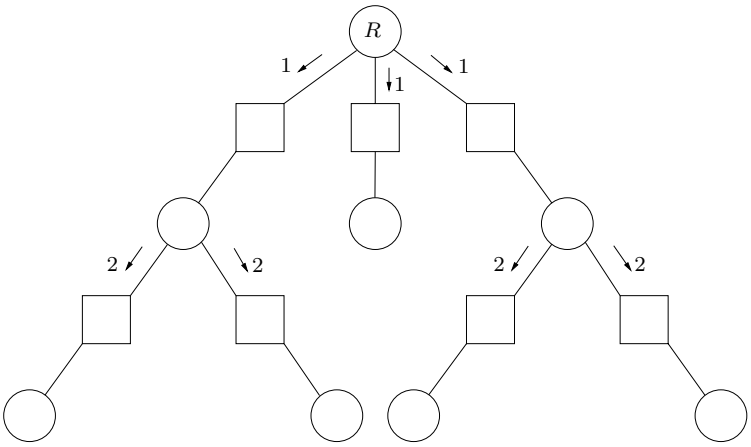


Fig. 4.21. The messages passing in DISTRIBUTEEVIDENCE(R).

Theorem 4.5. *Let the junction tree T represent the Bayesian network BN over the universe U and with evidence e. Assume that T is full.*

1. *Let V be a clique with set of potentials Φ_V, and let $S_1, \ldots, S_k$ be V's neighboring separators and with V-directed messages $\Psi_1, \ldots, \Psi_k$. Then*

$$P(V, e) = \prod \Phi_V \prod \Psi_1 \cdot \ldots \cdot \prod \Psi_k.$$

2. *Let S be a separator with the sets Ψ_S and Ψ^S in the mailboxes. Then*

$$P(S, e) = \prod \Psi_S \prod \Psi^S.$$

Proof.

1. Consider the messages passed in the direction of V. They correspond to a COLLECTEVIDENCE(V), and the message passing corresponds to a perfect elimination sequence ending with the nodes of V. Therefore,

$$P(V, e) = P(U, e)^{\downarrow V} = \prod \Phi_V \prod \Psi_1 \cdots \prod \Psi_k.$$

2. Consider S_k as before. Because

$$\prod \Psi^k = \left(\prod \Phi_V \prod \Psi_1 \cdots \prod \Psi_{k-1} \right)^{\downarrow S_k},$$

we have

$$\begin{aligned} P(S_k, e) = P(V, e)^{\downarrow S_k} &= \left(\prod \Phi_V \prod \Psi_1 \cdots \prod \Psi_k \right)^{\downarrow S_k} \\ &= \left(\prod \Phi_V \prod \Psi_1 \cdots \prod \Psi_{k-1} \right)^{\downarrow S_k} \prod \Psi_k \\ &= \prod \Psi^k \prod \Psi_k. \end{aligned}$$

□

4.5 Exploiting the Information Scenario

As mentioned at the beginning of this chapter, the actual information scenarios can provide simplifications of the calculations. This is one of the reasons why we let lazy propagation work with sets of potentials rather than multiplied potentials.

4.5.1 Barren Nodes

Barren nodes (see Section 4.1.2) do not contribute to the probabilities of nonbarren nodes, and therefore we need not take their potentials into account when calculating marginals of nonbarren nodes. This is illustrated in Figure 4.22.

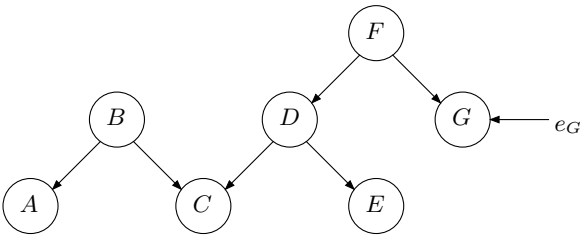


Fig. 4.22. The nodes A, B, C, D, and E are barren.

In the calculation of $P(F \mid e_G)$, the part of the network with barren nodes can be discarded. Figure 4.23 shows a junction tree for the network.

To calculate $P(F \mid e_G)$, you can collect to the clique (F, G). We see that all marginalizations to perform are of the form $\sum_X P(X \mid pa(X))$, and from the unit potential property (Section 1.4) they are all **1**.

Now assume that there is also evidence e_A for the variable A. Because A is d-separated from F, e_A does not affect $P(F \mid e_G)$. In the junction tree propagation, the message $\psi(B)$ from the clique (A, B) is no longer **1**. When the clique (B, C, D) produces a message for (D, F), the calculation is $\{P(C \mid B, D), P(B), P(a \mid B)\}^{\downarrow D}$. If we start marginalizing C out, we apply the unit potential property, and marginalizing B will result in a constant.

The handling of barren nodes can be taken care of using the following rule.

Barren node rule: Let Ψ be a set of potentials, and assume that we calculate $\Psi^{\downarrow V}$. If $A \notin V$, and the only potential in Ψ with A in the domain is of the form $P(A \mid W)$, then A is marginalized by discarding $P(A \mid W)$.

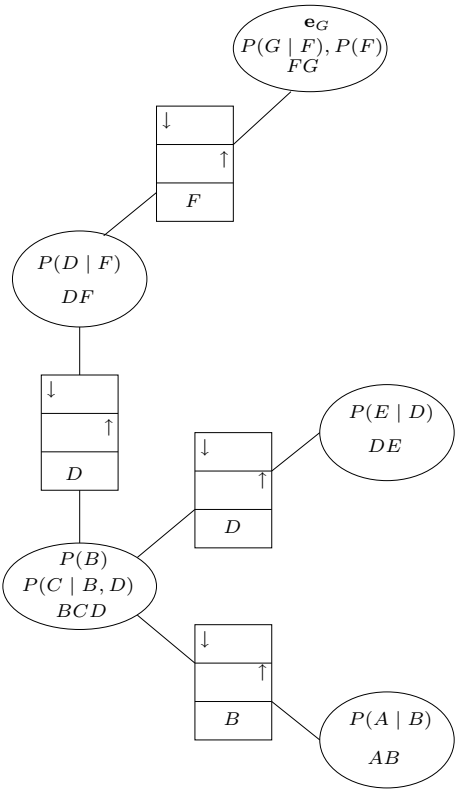


Fig. 4.23. A junction tree for the network in Figure 4.22.

4.5.2 d-Separation

When evidence is of the form that it instantiates a variable (hard evidence), then the domains to handle will be reduced with this variable. There are other simplifications due to instantiation: new pairs of variables may become d-separated, reducing the domains of the messages to communicate. We illustrate this with the example in Figure 4.24.

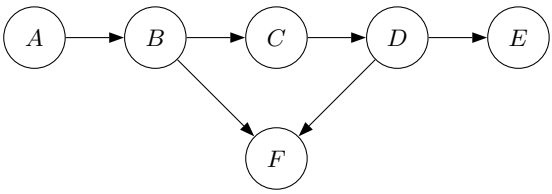


Fig. 4.24. A Bayesian network.

We will be interested in $P(E \mid e)$, and therefore we only consider collecting evidence to the clique (D, E). A junction tree for the network is shown in Figure 4.25.

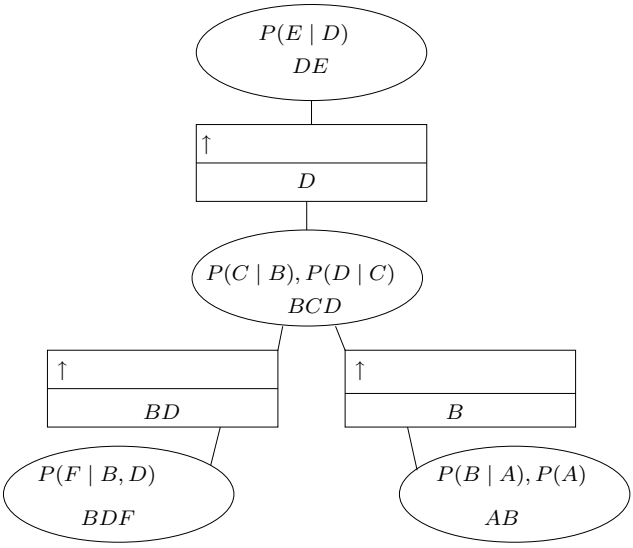


Fig. 4.25. A junction tree for the Bayesian network in Figure 4.24. Only the upward mailboxes are indicated.

First, let A be instantiated to a. The messages are given in Figure 4.26, and we see that the evidence has an impact on $P(E, e)$ through the message $\psi_1(D)$: $P(E \mid a) = \sum_D P(E \mid D)P(D \mid a)$.

Next, let C be instantiated to c. Then A and E are d-separated. Figure 4.27 shows how this is reflected in the messages: $P(E \mid c) = \sum_D P(E \mid D)\ P(D \mid c)$.

Finally, let F be instantiated to f. Then A and E are no longer d-separated. This is shown in Figure 4.28.

Note: In the examples, we have entered evidence on a variable X by instantiating the potentials including X. In general, evidence can be entered by adding the corresponding evidence potential to a clique containing X, and the instantiation is effected when X must be marginalized. This means that the evidence potential is passed to separators containing X.

4.6 Nontriangulated Domain Graphs

So far, we have considered propagation methods only for potentials with a triangulated graph. For these methods, we know that the junction tree is a

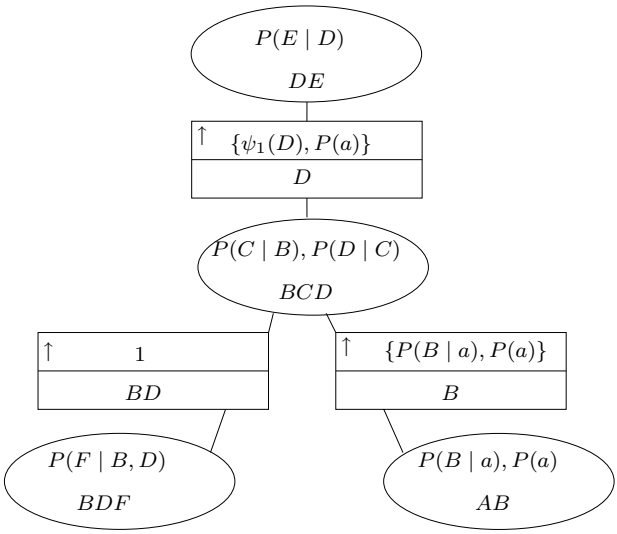


Fig. 4.26. The messages on collecting to (D, E) for A instantiated. Here $\psi_1 = \sum_C P(D \mid C) \sum_B P(C \mid B) P(B \mid a) = \sum_C P(D \mid C) P(C \mid a) = P(D \mid a)$.

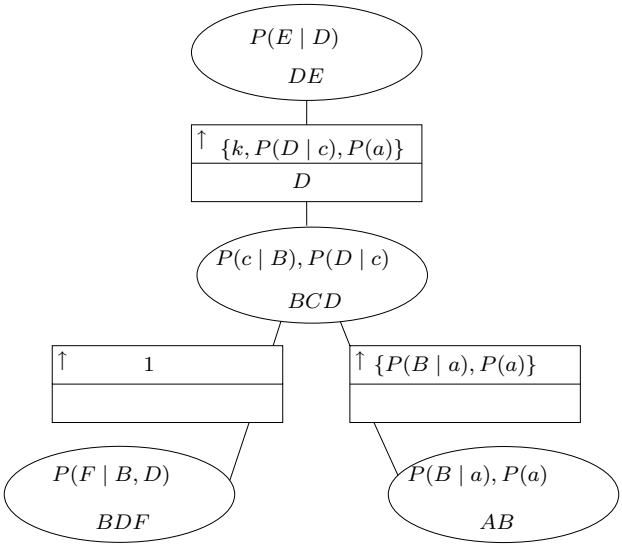


Fig. 4.27. The messages on collecting to (D, E) for A and C instantiated. Here $k = \sum_B P(c \mid B) P(B \mid a) = P(c \mid a)$.

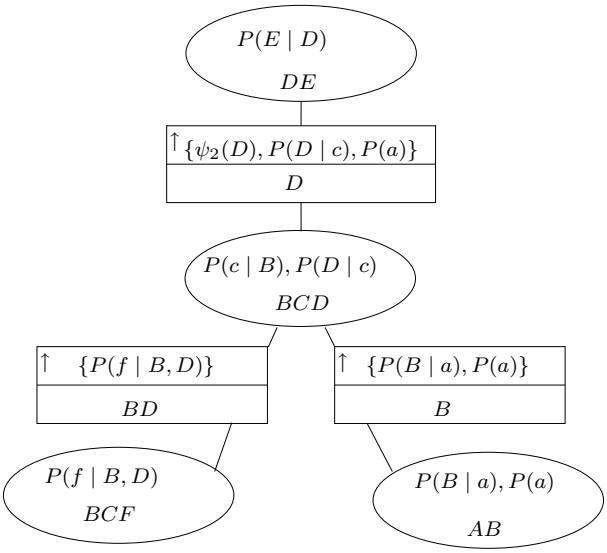


Fig. 4.28. The messages on collecting to (D, E) for A, C, and F instantiated. Here $\psi_2(D) = \sum_B P(f \mid B, D)(c \mid B)P(B \mid a)$.

propagation framework having the smallest possible domains with which to work.

If the domain graph is not triangulated, we embed it in a triangulated graph and use its junction tree. In fact, we did so in Section 4.5.2 when we handled evidence.

Example 4.4. Consider the Bayesian network in Figure 4.29. After having eliminated the variables A, C, H, I, and J, we cannot eliminate any node without adding fill-ins, and the graph is not triangulated.

The graph in Figure 4.30 is a triangulated graph extending the moral graph in Figure 4.29. We can use a junction tree for that graph (see Figure 4.31).

4.6.1 Triangulation of Graphs

It is quite easy to find a triangulated graph extending a graph G. You eliminate the variables in some order, and if you wish to eliminate a node with an incomplete neighbor set, you make it complete by adding fill-ins (the graph in Figure 4.30 is the result of eliminating in the order $A, C, H, I, J, B, G, D, E, F$). The resulting graph has a perfect elimination sequence, and it is therefore triangulated.

There are several different elimination orders, and many of them produce different triangulated graphs. We aim to work with the one yielding the smallest domains.

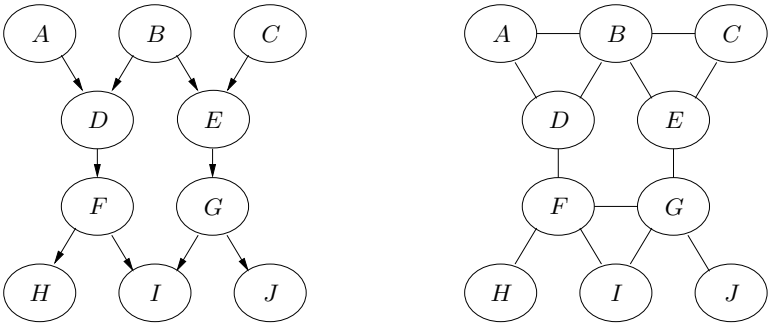


Fig. 4.29. A Bayesian network (a) with a nontriangulated moral graph (b).

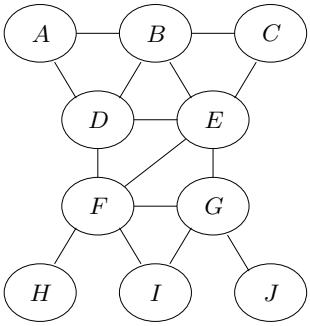


Fig. 4.30. A triangulated graph extending the moral graph in Figure 4.29.

Definition 4.8. *Let $\mathcal{V}$ be a set of variables. For $X \in \mathcal{V}$, $|\mathrm{sp}(X)|$ denotes the number of states of X. The* size of $\mathcal{V}$, $\mathrm{sz}(\mathcal{V})$, *is the product $\prod_{X \in \mathcal{V}} |\mathrm{sp}(X)|$. Let BN be a Bayesian network, let G be a triangulated graph extending BN's moral graph, and let $V_1, \ldots, V_n$ be the cliques of G. The* size of G *is the sum* $\mathrm{size}(G) = \sum_i \mathrm{sz}(V_i)$.

Unfortunately, it is NP-hard to determine an elimination sequence yielding a triangulation of minimal size. However, there are heuristic algorithms that have proven to give fairly good results. One example is the following:

Heuristic: Repeatedly eliminate a simplicial node, and if this is not possible, eliminate a node X of minimal $\mathrm{sz}(\mathrm{fa}(X))$.

Example 4.5. Let the number of states for the variables in Figure 4.29 be as follows: A, B, C, H, I, and J have two states, D has four states, E has five states, F has six states, and G has seven states. After having eliminated the variables A, C, H, I, and J, we eliminate a nonsimplicial node. We have $\mathrm{sz}(\mathrm{fa}(B)) = 40, \mathrm{sz}(\mathrm{fa}(D)) = 48, \mathrm{sz}(\mathrm{fa}(E)) = 70, \mathrm{sz}(\mathrm{fa}(F)) = 168$, and

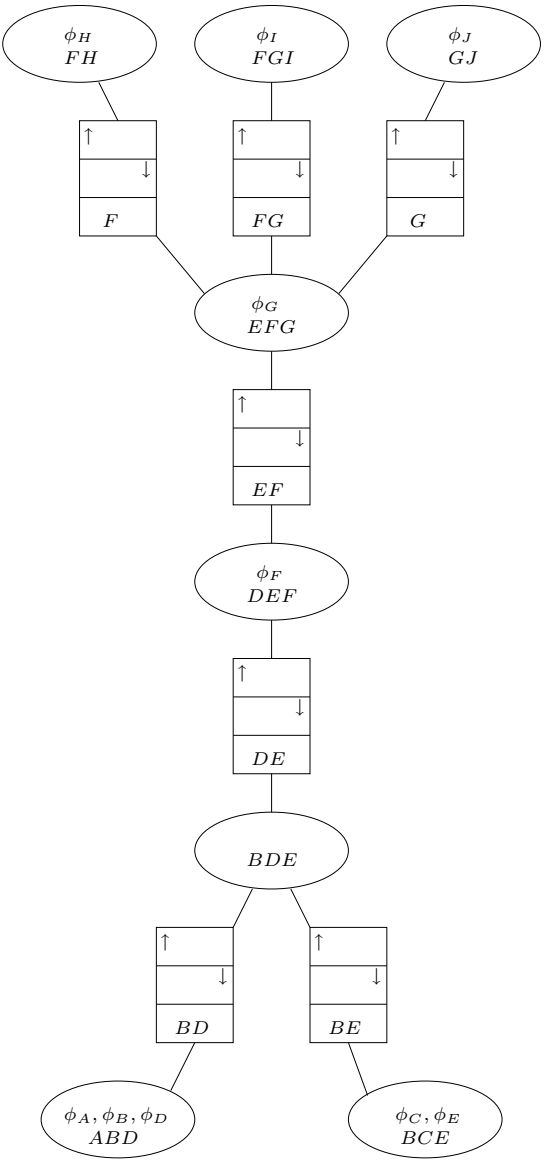


Fig. 4.31. A junction tree with potentials from the Bayesian network in Figure 4.29. Notation: $\phi_X = P(X \mid pa(X))$.

$\text{sz}(\text{fa}(G)) = 210$. We choose to eliminate B, creating the fill-in (D, E). With this new link, we have new sizes $\text{sz}(\text{fa}(D)) = 120$ and $\text{sz}(\text{fa}(E)) = 140$. We eliminate D and add the fill-in (E, F). Now the graph is triangulated. However, in this case the triangulation is not optimal (see Exercise 4.32).

For later use, we establish the following proposition.

Proposition 4.6. *Let $A_1, \ldots, A_n$ be an elimination sequence triangulating the graph G, and let A_i and A_j be two nonneighbors in $G (i < j)$. Then the elimination sequence introduces the fill-in (A_i, A_j) if and only if there is a path $A_i - X - \cdots - A_j$ such that all intermediate nodes are eliminated before A_i.*

Proof. Assume that fill-ins may be introduced that violate the proposition, and let (A_i, A_j) be such a fill-in with i as small as possible. Let the link be introduced on eliminating the node A_k. Because new fill-ins cannot be attached to A_i when it has been eliminated, we must have $k < i$. One of the links (A_k, A_j) and (A_i, A_k) on eliminating A_k must be a fill-in (if not, the (A_i, A_j) fill-in does not violate the proposition). Let it be (A_i, A_k). Due to the choice of (A_i, A_j) the proposition holds for (A_i, A_k), hence there is a path $A_k - X - \cdots - A_i$ such that all intermediate nodes are eliminated before A_k (see Figure 4.32). If also (A_k, A_j) is a fill-in, the same must hold. Connecting these two paths yields a path $A_i - X - \cdots - A_j$ such that all intermediate nodes are eliminated before A_i, a contradiction.

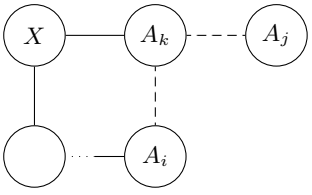


Fig. 4.32. A path connecting A_i and A_j through nodes eliminated before A_i.

Next, assume that we have a path $A_i - X - \ldots - A_j$ such that all intermediate nodes are eliminated before A_i. Let A_k be any node on the path to be eliminated, and let Y and Z be the neighbors on the path. After the elimination of A_k, there is a link (Y, Z), and there is still a path $A_i - X - \ldots - A_j$ such that all intermediate nodes are eliminated before A_i, so the property is invariant under elimination. When all the nodes before A_i are eliminated, the path must be the link (A_i, A_j). □

4.6.2 Triangulation of Dynamic Bayesian Networks

Return to Exercise 3.25 and consider the model in Figure 3.52. In Figure 4.33, we have folded it out to three time slices.

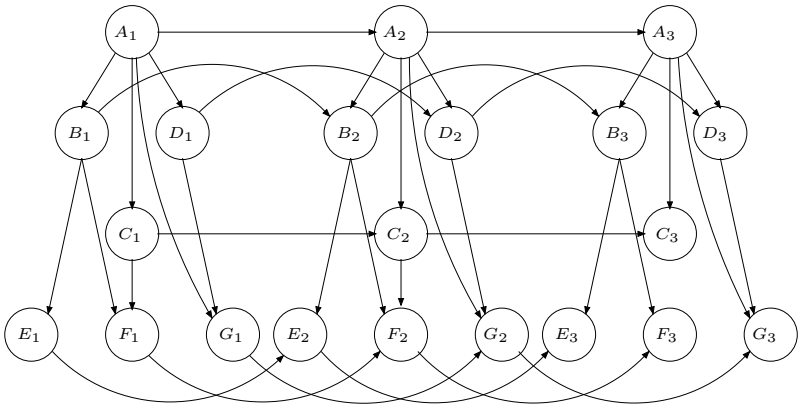


Fig. 4.33. Three time slices of the model in Figure 3.52.

As you have probably experienced when solving Exercise 3.25, your computer ran out of memory when you tried to compile the model folded out to four or five time slices. The reason is that the cliques become too large.

A conceptually simple way of considering propagation in dynamic Bayesian network models is that information is transmitted from one time slice to the next (if the task is forecasting) or to the previous time slice (if the task is to find out what happened in the past). In other words, probability potentials describing time slice i are transmitted from time slice i to time slice $i+1$ or to time slice $i-1$.

Let us consider forward passing from time slice i to time slice $i+1$, and let W be the set of variables with a child in slice $i+1$. We wish to pass potentials representing the joint probability of W. For the model in Figure 4.33, we pass the information from slice 1 to slice 2 by eliminating all nodes in slice 1 before any node from slice 2 is eliminated. Now consider any pair of nodes (X_2, Y_2). If there is a path in slice 1 connecting them, then they will be linked after the elimination of slice 1 (Proposition 4.6). Because the moral graph for slice 1 is connected, and all nodes in slice 2 have a parent in slice 1, the entire slice 2 will be a subset of a clique if slice 1 is eliminated before any node from slice 2. If you process only two time slices, you may avoid this clique explosion by using another elimination sequence. However, it will inevitably arrive when you extend the number of time slices to process. Some cliques will contain all variables with a child in the next slice or will contain all variables with a parent in the previous slice.

This situation is not reserved for models with connected time slices. Consider the model in Figure 4.34. If the model is folded out to four time slices, and the first three slices are eliminated before any node from slice four, then slice four becomes a complete set. Figure 4.35 shows the moral graph for four

slices of the model. The reader can check that all pairs of nodes in slice four have a connecting path through the past slices.

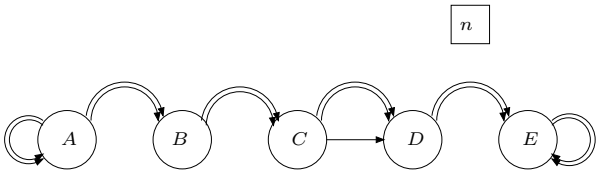


Fig. 4.34. A dynamic Bayesian network model with very sparse connection inside the time slices.

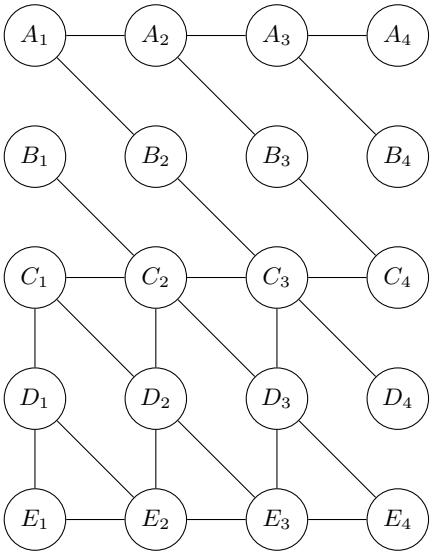


Fig. 4.35. The moral graph for four time slices of the model in Figure 4.34.

As indicated above, you may think of propagation in dynamic Bayesian networks as a way of passing probabilities of output nodes forward in time. The problem is that most often, the required probability distribution is the joint distribution over all output variables. If this is intractable, you can approximate the joint distribution by partitioning the set of output variables. If $\mathcal{O}$ is partitioned into $\{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$, then instead of passing $P(\mathcal{O})$ you pass $\{P(\mathcal{O}_1), P(\mathcal{O}_2), P(\mathcal{O}_3)\}$. It has been proven that the error introduced does not accumulate over time, but converges to a finite error (in Kullback-Leibler divergence; see Definition 6.2).

4.7 Exact Propagation with Bounded Space

One of the biggest problems with exact propagation algorithms such as the junction tree based approach described in Section 4.4 is that the probability tables can become intractably large. In this section we will investigate an exact propagation algorithm in which space can be traded for time. For this particular propagation algorithm, we will consider calculation of probabilities only of the form $P(x, \mathbf{e})$, since $P(x \mid \mathbf{e})$ can subsequently be found by $P(x \mid \mathbf{e}) = P(x, \mathbf{e})/(\sum_x P(x, \mathbf{e}))$.

4.7.1 Recursive Conditioning

Consider the Bayesian network in Figure 4.36 and assume that we are interested in the probability $P(f)$.

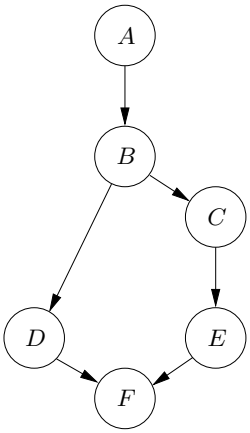


Fig. 4.36. A Bayesian network.

By calculating $P(f)$ using, for example, variable elimination (Section 2.3.4) or lazy propagation (Section 4.4.1), we basically first establish an elimination sequence and then use the distributive law. For example, by using the elimination sequence F, E, D, C, B, A we would get

$$\begin{aligned} & P(f) \\ & \quad = \sum_A \sum_B \sum_C \sum_D \sum_E P(A, B, C, D, E, f) \\ & \quad = \sum_A P(A) \sum_B P(B \mid A) \sum_C P(C \mid B) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E). \end{aligned} \tag{4.1}$$

The sequence in which the calculations are performed can be encoded in a *computation tree*. The leaf nodes represent the conditional probability distributions in the model, and for each internal node the potentials defined by the two subtrees are multiplied and the variables indicated by the label of the node are marginalized out (see Figure 4.37).

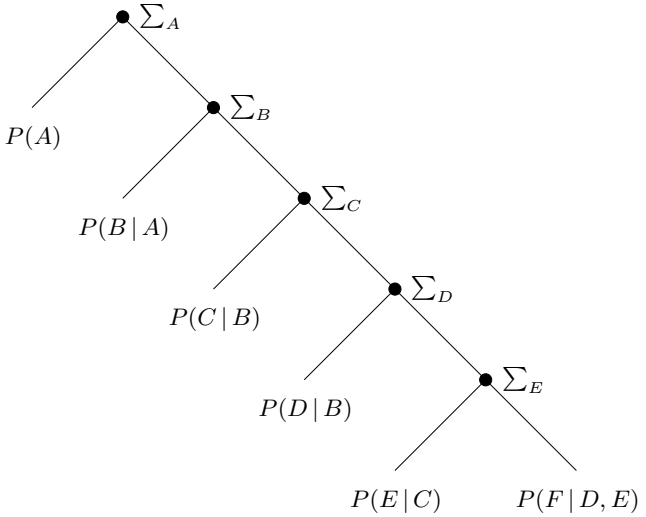


Fig. 4.37. The computation tree for the calculation of $P(f)$ in Figure 4.36 using the elimination sequence E, D, C, B, A.

Based on the computation tree in Figure 4.37 we can easily specify an algorithm that calculates $P(f)$ and performs the same operations as in equation (4.1): evaluate the computation tree from the leaves toward the root. When an internal node is reached, multiply the two potentials calculated in the two subtrees for that node and marginalize out the appropriate variables.

Another way of doing the calculations would be to start at the root $\sum_A$ and recursively evaluate the subtrees for each state of A; when the recursive calls return, the results are added up. Assuming that A is binary, for the calculations in equation (4.1) this would correspond to

$$\begin{aligned} P(f) = {} & P(a_1) \sum_B P(B \mid a_1) \sum_C P(C \mid B) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E) \\ & + P(a_2) \sum_B P(B \mid a_2) \sum_C P(C \mid B) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E), \end{aligned} \tag{4.2}$$

where, for example, the first term is the result of the recursive calls made at node $\sum_B$:

$$\begin{aligned}\sum_B P(B \mid a_1) \sum_C P(C \mid B) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E) \\ = P(b_1 \mid a_1) \sum_C P(C \mid b_1) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E) \\ + P(b_2 \mid a_1) \sum_C P(C \mid b_2) \sum_D P(D \mid C) \sum_E P(E \mid D) P(f \mid D, E).\end{aligned}$$

Compared to equation (4.1) we can say that when the computation tree is "read" from the root toward the leaves, we condition in the internal nodes, and when it is "read" from the leaves towards the root, we marginalize out in the internal nodes.

By continuing the "recursive conditioning" above, we see that the storage requirements are considerably reduced. Specifically, for handling the intermediate results we have to store only the initial conditional probability distributions together with a single number for each internal node in the computation tree, i.e., the space complexity is linear in the number of nodes. Unfortunately, this reduction in space comes at a price. In this particular example, the number of recursive calls corresponds to the size of the state space of all the variables involved. Assuming that the variables are binary, this would amount to 32 recursive calls. Note, however, that the size of the call stack is proportional to the depth of the tree.

In general, the number of recursive calls increases exponentially with the height of the computation tree, so to reduce the time complexity we should aim for a more balanced tree structure. For example, consider again the Bayesian network in Figure 4.36, but assume now that we have the elimination ordering B, A, E, C, D:

$$\begin{aligned}P(f) &= \sum_B \sum_A \sum_E \sum_C \sum_D P(A, B, C, D, E, f) \\ &= \sum_B \left[\sum_A P(A) P(B \mid A) \right] \\ &\quad \times \left[\sum_E \left[\sum_C P(C \mid B) P(E \mid C) \right] \left[\sum_D P(D \mid B) P(f \mid D, E) \right] \right].\end{aligned} \tag{4.3}$$

The corresponding computation tree is shown in Figure 4.38. In this tree the calculation of $P(f)$ requires only $2 \cdot (2 + 2 \cdot (2 + 2)) = 20$ recursive calls.

In the two examples above, we condition on only one variable at a time. The reason is that both elimination sequences ensure that each time we condition on a variable, the remaining variables can be partitioned into two d-separated sets. This, however, is not the case in general. For example, for the elimination sequence D, C, E, B, A, neither D nor C can alone partition the variables into independent sets; hence a node in the tree is labeled with both variables (see Figure 4.39):

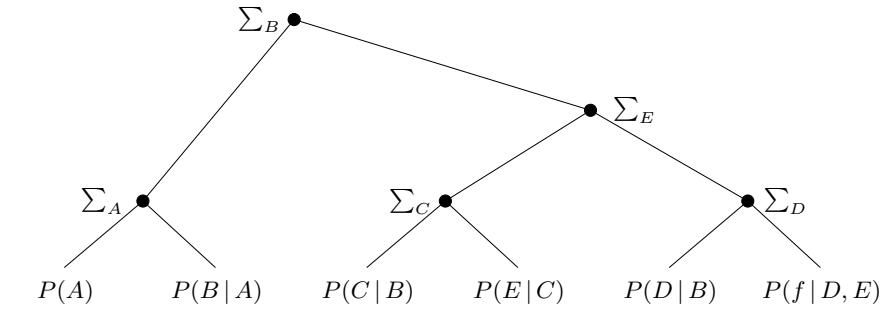


Fig. 4.38. The computation tree for the calculation of $P(f)$ in Figure 4.36 using the elimination sequence B, A, E, C, A.

$$
\begin{aligned}
P(f) &= \sum_D \sum_C \sum_E \sum_B \sum_A P(A, B, C, D, E, f) \\
&= \sum_D \sum_C \left[\sum_E P(E \mid C) P(f \mid D, E) \right] \\
&\quad \times \left[\sum_B P(C \mid B) P(D \mid B) \left[\sum_A P(A) P(B \mid A) \right] \right].
\end{aligned}
$$

It should also be noted that the computation graph is not required to be binary; for example, if conditioning on a variable partitions the remaining variables into three or more d-separated sets, then the corresponding node may have more than two children in the computation tree.

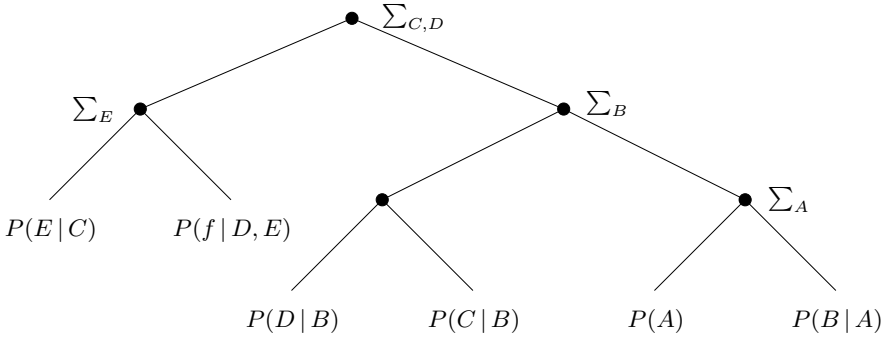


Fig. 4.39. The computation tree for the calculation of $P(f)$ in Figure 4.36 using the elimination sequence D, C, E, B, A.

In general, the set of variables attached to a node T corresponds to the set of noninstantiated variables shared by its two subtrees T_l and T_r. This set is also called the *cutset* for the node:

$$\text{cutset}(T) = (\text{dom}\,(T_l) \cap \text{dom}\,(T_r)) \setminus \text{a-cutset}(T),$$

where $\text{dom}\,(T_i)$ are the variables that appear in the conditional probability tables in the subtree T_i and a-cutset(T) is the union of the cutsets associated with the ancestral nodes for T in the tree (if T is the root node, then a-cutset$(T) = \emptyset$). Thus, the a-cutset is the set of nodes already instantiated. For example, in the tree in Figure 4.39, the cutset for the root node is $\{C, D\}$ and the a-cutset for the node labeled $\sum_A$ is $\{B, C, A\}$. In particular, the a-cutset for the unlabeled node is $\{B, C, D\}$, which covers all variables in the subtree, hence this node is given the empty cutset.

Before we present a more-formal specification of the algorithm it should be noted that in the above examples we incorporated the evidence f directly in the computation tree, indicating that a new computation tree is constructed for each piece of evidence. A more-efficient approach would be to first construct a single computation tree with no evidence inserted. Then, when evidence arrives we simply "record" the variables that should be instantiated such that no summations are performed for these variables.

Algorithm 4.2 reflects this approach for calculating the probability of a configuration $\mathbf{e}$ based on a computation tree for a Bayesian network. Observe that at each recursive call we record the corresponding instantiation and unrecord it when the call returns.

Algorithm 4.2 [RecursiveConditioning] *In order to calculate $P(\mathbf{e})$ using recursive conditioning on the tree T, do:*

1. *If T is the root, then record instantiation $\mathbf{e}$.*
2. *If T is a leaf, then:*
   a) *Return LookUp(T).*
3. *Else*
   a) *Set $p := 0$.*
   b) *For each noninstantiated configuration $\mathbf{c}$ of cutset(T) do:*
      i. *Record instantiation $\mathbf{c}$.*
      ii. *Set*
      $$p := p + \prod_{i=1}^{m} \mathit{RecursiveConditioning}(T_i),$$
      where $T_1, \ldots, T_m$ are the children of T.
      iii. *Unrecord instantiation $\mathbf{c}$.*
   c) *Return p.*

□

Algorithm 4.3 [LookUp] *To find the value of the leaf node T under the recorded instantiations, do:*

1. *Let X be the variable associated with T and let $P(X \mid \text{pa}(X))$ be the conditional probability table assigned to X.*
2. *If X is instantiated, then:*

a) Let x be the recorded instantiation for X and let π be the recorded instantiation for $\mathrm{pa}(X)$.
b) Return $P(x \mid \pi)$.
3. else
a) Return 1.

□

Clearly, this algorithm requires only as much space as is needed to store the computation tree, and this is linear in the number of variables (hence for this aspect the shape of the tree is of no importance). However, the situation is different if we consider the time complexity. The time complexity can be estimated by counting the number of recursive calls, and it can be shown (see Exercise 4.38) that for a balanced tree it is $O(n^{w+1})$ and for an unbalanced tree it is $O(n \cdot \exp(w \cdot n)))$, where w is the size of the largest cutset.

This also indicates that it is important to find a good computation tree representation of the Bayesian network, and as we also indicated above this is closely connected with finding a good elimination sequence (see Section 4.6.1). In fact, given an elimination sequence that produces a maximum clique size of w, there are algorithms that will return a computation tree in which the cutset is not larger than w. The idea is to build the tree from the leaves to the root, where appropriate subtrees are joined according to the sequence in which the variables are marginalized out.

As for the tree in Figure 4.37, the algorithm above may perform redundant recursive calls to a subtree. This may happen when the a-cutset for a node/subtree includes a variable that is not in the domain of any of the probability tables associated with the subtree in question; we shall call all nonredundant nodes in a-cutset(T) the *context* for T. A way of controlling the number of redundant recursive calls is to cache previous calculations. Since we assume that we do not have enough memory to cache all values, the trick is therefore to find a good strategy for selecting the values to cache. If cache?$_T(\mathbf{x})$ is a function that determines whether to cache the value for subtree T evaluated in the context $\mathbf{x}$, we can directly control how much memory the algorithm is allowed to use. Algorithm 4.2 can easily be modified to support such a caching strategy: before a recursive call is made in context $\mathbf{x}$ we check whether a value for that context is already stored in the cache; if this is the case we simply return that value; otherwise, the call is completed and the result is cached if this is in accordance with cache?$_T(\mathbf{x})$.

4.8 Stochastic Simulation in Bayesian Networks

The junction tree based propagation methods described in the beginning of this chapter require tables for the cliques in the triangulated graph. These cliques may be very large, and it may happen that the space requirements of the tables cannot be met by the hardware available. When this is the case

either you can make a tradeoff between time and space (using, for example, recursive conditioning as described in Section 4.7) or you can trade space for accuracy by using an approximate inference method.

In this section, we give a flavor of a class of approximate methods that are based on a technique called *stochastic simulation*. To illustrate the methods, consider the Bayesian network in Figure 4.40, with the conditional probabilities specified in Table 4.1, and assume that we want to estimate the probability of $E = y$. Now suppose also that we have access to a database containing configurations over the five variables and for which the distribution of the configurations follows the probability distribution specified by the Bayesian network. Given such a database, we can estimate the probability of $E = y$ by counting the number of cases that contain $E = y$ and divide it by the total number of cases:

$$P(E) \approx \frac{N(E = y)}{N}.$$

Since we (usually) do not have access to such a database, stochastic simulation instead tries to simulate such an access. This is done by drawing a large number of random configurations over (A, B, C, D, E) using the Bayesian network. There are several different algorithms for performing this type of sampling, and their main differences lie in how the samples are generated and how the probabilities are estimated from the sampled configurations.

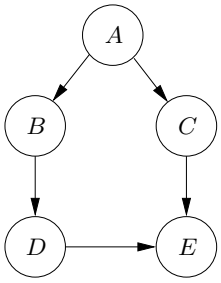


Fig. 4.40. An example network. All variables have the states y and n.

4.8.1 Probabilistic Logic Sampling

Probabilistic logic sampling is one of the simplest sampling procedures. To illustrate the approach, consider again the Bayesian network in Figure 4.40 and assume for simplicity that we have not received any evidence. A configuration can now be sampled by iteratively sampling a state of each of the variables. First a state of variable A is sampled. A random generator (with an even distribution) is asked to give a real number between 0 and 1. If the number is less than 0.4 (the prior probability of $A = y$), the state is y; otherwise, the

B	A: y	A: n
y	0.3	0.8
n	0.7	0.2

$P(B \mid A)$

C	A: y	A: n
y	0.7	0.4
n	0.3	0.6

$P(C \mid A)$

D	B: y	B: n
y	0.5	0.1
n	0.5	0.9

$P(D \mid B)$

D	C: y	C: n
y	(0.9, 0.1)	(0.999, 0.001)
n	(0.999, 0.001)	(0.999, 0.001)

$P(E \mid C, D)$

Table 4.1. The conditional probabilities for the example network. $P(A) = (0.4, 0.6)$.

state is n. Assume that the result is y. From the conditional probability table $P(B \mid A)$, we have that $P(B \mid y) = (0.3, 0.7)$. The random generator is asked again, and if the number is less than 0.3, the state of B is y. This procedure continues until we also have a state for C, D, and E. Observe that the sequence in which we generate the sample follows the topological ordering of the nodes in the network: we start at the nodes without parents and work ourselves toward the nodes without children; when visiting a variable we sample a state for that variable using its associated probability table conditioned on the configuration of the parent variables that have already been sampled.

The next configuration is sampled through the same procedure, and this is repeated until N configurations have been sampled. In Table 4.2, an example set of configurations is given.

AB	CDE: yyy	yyn	yny	ynn	nyy	nyn	nny	nnn
yy	4	0	5	0	1	0	2	0
yn	2	0	16	0	1	0	8	0
ny	9	1	10	0	14	0	16	0
nn	0	0	4	0	0	0	7	0

Table 4.2. A set of 100 configurations of (A, B, C, D, E) sampled from the network in Figure 4.40 and Table 4.1.

The probability distributions for the variables can now be calculated by counting in the sample set (see Exercise 4.39). For example, for 99 of the samples in Table 4.2, the state of E is y, and this gives an estimated probability:

$$P(E) \approx \left(\frac{N(E = y)}{N}, \frac{N(E = n)}{N} \right) = \left(\frac{99}{100}, \frac{1}{100} \right) = (0.99, 0.01).$$

So far, only marginal probabilities have been calculated. However, a straightforward approach to handle evidence is simply to discard the configurations that do not conform to it. In other words, a new series of stochastic simulations is started, and whenever a state of an observed variable is drawn, you stop simulating if the state drawn is not the one observed. In general, if we have evidence $\mathbf{e}$ and we are interested in estimating $P(X_k \mid \mathbf{e})$ using N samples, then probabilistic logic sampling can be performed as follows:

1. Let $(X_1, \ldots, X_n)$ be a topological ordering of the variables.
2. For $j = 1$ to N:
   a) For $i = 1$ to n:
      - Sample a state x_i for X_i using $P(X_i \mid \text{pa}(X_i) = \pi)$, where π is the configuration already sampled for $\text{pa}(X_i)$.
   b) If $\mathbf{x} = (x_1, \ldots, x_n)$ is consistent with $\mathbf{e}$, then
   $$N(X_k = x_k) := N(X_k = x_k) + 1,$$
   where x_k is the state that was sampled for X_k.
3. Return:
$$P(X_k = x_k \mid \mathbf{e}) \approx \frac{N(X_k = x_k)}{\sum_{x \in \text{sp}(X_k)} N(X_k = x)}.$$

The preceding method does not require a triangulation of the network, nor is it necessary to store the sampled configurations (as we did in Table 4.2). It is enough to store the counts for each variable of interest. Whenever a sampled configuration has been determined, the counts of all variables are updated, and the sample can be discarded. The method therefore saves much space, and each configuration is determined in time linear in the number of variables. These benefits, however, come at the expense of accuracy and time. In particular, this method has a serious drawback when the probability of the evidence is small. For instance, assume that for the preceding example we have the observations $B = n$ and $E = n$. The probability for $(B = n, E = n)$ is 0.00282, which means that in order to get 100 configurations, you should for this tiny example expect to perform more than 35,000 stochastic simulations. In general, since the probability of the evidence drops off exponentially fast, this method can be hopelessly time-consuming even when we have only a few pieces of evidence.

4.8.2 Likelihood Weighting

You might be tempted to overcome the shortcoming of probabilistic logic sampling by simply fixing the evidence variables $\mathcal{E}$ to their observed states and sample only from the nonevidence variables; in this way no samples need to be discarded. However, since a sample is generated by going from the root nodes down to the leaves, this naive procedure would result in a sample in which the value for a given variable takes only the evidence from its ancestors

into account and not the evidence pertaining to the variables further down in the network. For example, if we should try to estimate $P(A \mid B = n, E = n)$ using this modified sampling procedure we would actually estimate $P(A)$. The problem is that instead of sampling from the distribution $P(\mathcal{U}, \mathbf{e})$ specified by the evidence and the Bayesian network, we are in fact sampling from a probability distribution somewhere in between the prior distribution $P(\mathcal{U})$ and the posterior distribution $P(\mathcal{U} \mid \mathbf{e})$. To be more precise, if $\mathrm{pa}(X)''$ are the parents of X that have received evidence ($\mathrm{pa}(X) = \mathrm{pa}(X)' \cup \mathrm{pa}(X)''$), then the joint distribution $P(\mathcal{U}, \mathbf{e})$ that we would like to sample from can be expressed as

$$P(\mathcal{U}, \mathbf{e}) = \underbrace{\prod_{X \in \mathcal{U} \setminus \mathcal{E}} P(X \mid \mathrm{pa}(X)', \mathrm{pa}(X)'' = \mathbf{e})}_{\text{Part 1}} \times \underbrace{\prod_{X \in \mathcal{E}} P(X = e \mid \mathrm{pa}(X)', \mathrm{pa}(X)'' = \mathbf{e})}_{\text{Part 2}}. \tag{4.4}$$

However, the distribution that we are actually sampling from is

$$\text{Sampling distribution} = \prod_{X \in \mathcal{U} \setminus \mathcal{E}} P(X \mid \mathrm{pa}(X)', \mathrm{pa}(X)'' = \mathbf{e}),$$

which corresponds only to Part 1 of equation (4.4).

Fortunately, this also points to a simple way of compensating for the estimation problem above: weigh each of the generated samples $\mathbf{x}$ with a weight corresponding to Part 2 of equation (4.4). That is, instead of adding 1 to the count $N(X_i)$ (as we did for probabilistic logic sampling) we add a weight $w(\mathbf{x}, \mathbf{e})$:

$$w(\mathbf{x}, \mathbf{e}) = \prod_{E \in \mathcal{E}} P(E = e \mid \mathrm{pa}(X) = \pi),$$

where π is the configuration of $\mathrm{pa}(X)$ specified by $\mathbf{x}$ and $\mathbf{e}$.

This updating approach, called *likelihood weighting*, ensures that we get the correct counts for estimating the probabilities. This can also be seen by combining the weight calculation and the sampling distribution, which together correspond to the distribution $P(\mathcal{U}, \mathbf{e})$.

Now consider again the example network above and assume that we want to estimate $P(A \mid B = n, E = n)$. As before, we start by sampling a state of A using a random generator (let the resulting state be y). Since B has received the evidence $B = n$, no state is sampled, and instead we continue to C and sample a state using $P(C \mid A = y) = (0.7, 0.3)$; assume that the sampled state is n. Next we sample a state for D using $P(D \mid B = n) = (0.5, 0.5)$ (assume that we get $D = y$). Since E has received evidence, $E = n$, we now have a complete configuration over all five variables and the sampling stops. Next we calculate the weight associated with the sampled configuration:

$$w((A = y, B = n, C = n, D = y, E = n), (B = n, E = n))$$
$$= P(B = n \mid A = y)P(E = n \mid C = n, D = y) = 0.7 \cdot 0.001 = 0.0007.$$

This value is then added to $N(A = y)$ (and to $N(C = n)$ and $N(D = y)$ as well if we are also interested in the probabilities for these two variables). We then continue to generate samples (and weights) as above, and when a sufficient number of samples has been generated we return the estimate $P(A \mid B = n, E = n) \approx N(A = y)/(N(A = y) + N(A = n))$.

In general, if we are interested in estimating $P(X_k \mid \mathcal{E} = \mathbf{e})$ using N samples, then the likelihood weighting algorithm can be summarized as follows:

1. Let $(X_1, \ldots, X_n)$ be a topological ordering of the variables.
2. For $j = 1$ to N:
   a) w:=1.
   b) For $i = 1$ to n:
      - Let $\mathbf{x}'$ be the configuration of $(X_1, \ldots, X_{i-1})$ specified by $\mathbf{e}$ and the previous samples.
      - If $X_i \notin \mathcal{E}$, then:
        - Sample a state x_i for X_i using $P(X_i \mid \text{pa}(X_i) = \pi)$, where $\text{pa}(X_i) = \pi$ is consistent with $\mathbf{x}'$.
      - else
        $w := w \cdot P(X_i = e_i \mid \text{pa}(X_i) = \pi)$, where $\text{pa}(X_i) = \pi$ is consistent with $\mathbf{x}'$.
   c) $N(X_k = x_k) := N(X_k = x_k) + w$, where x_k is the sampled state for X_k.
3. Return:
$$P(X_k = x_k \mid \mathbf{e}) \approx \frac{N(X_k = x_k)}{\sum_{x \in \text{sp}(X_k)} N(X_k = x)}.$$

Although likelihood sampling is an improvement over probabilistic logic sampling it may still require a large number of samples. This is typically the case when there is a large difference between the sampling distribution and $P(\mathcal{U}, \mathbf{e})$ and, again, this is often the case when the evidence is unlikely.

4.8.3 Gibbs Sampling

Other methods have been constructed for dealing with this problem. A widely used method is *Gibbs sampling*. In Gibbs sampling, you start with some configuration consistent with the evidence (for example determined by probabilistic logic sampling), and then you randomly change the state of the variables in topological order. In one sweep through the variables, you determine a new configuration, and then you use this configuration for a new sweep, and so on. From this perspective, Gibbs sampling differs from the above two procedures by generating a new sample based on the current one.

Consider again the example above and let the evidence be $B = n$ and $E = n$. Assume also that we are given the starting configuration *ynyyn*. Now,

to generate a sample we first calculate the probability of A given the other states of that configuration, that is, $P(A \mid B = n, C = y, D = y, E = n)$. From the network, we see that the Markov boundary for A includes only B and C; hence it is sufficient to calculate $P(A \mid B = n, C = y)$. It is easily done by Bayes' rule, which gives (0.8, 0.2). We then draw a number from the random generator, and let us assume that the number is 0.456, resulting in $A = y$. The next free variable is C. We calculate

$$\begin{aligned} P(C \mid A = y, B = n, D = y, E = n) &= P(C \mid A = y, D = y, E = n) \\ &= (0.996, 0.04), \end{aligned}$$

and draw a number from the random generator; assume that it results in $C = y$.

In general, the calculation proceeds as follows. Let A be a variable in a Bayesian network BN, let $B_1, \ldots, B_n$ be the remaining variables, and let $\mathbf{b} = (b_1, \ldots, b_n)$ be a configuration of $(B_1, \ldots, B_n)$. Then, $P(A, \mathbf{b})$ is the product of all conditional probabilities in BN with B_i instantiated to b_i. Therefore, $P(A, \mathbf{b})$ is proportional to the product of the potentials involving A, and $P(A \mid \mathbf{b})$ is the result of normalizing this product. Note that the calculation of $P(A \mid \mathbf{b})$ is a local task.

To return to the example, the next variable is D, and we follow the same procedure. Assume that the result is $D = n$. Then the configuration from the first sweep is *ynynn*. The next sweep follows the same procedure. Assume that the state of A changes to n. Then we calculate $P(C \mid A = n, D = n, E = n)$ and so forth.

In this way, a large sample of configurations consistent with the observations is produced. The question is whether the sample is representative of the probability distribution. It is not always so. It may be that the initial configuration is rather improbable, and therefore the first samples likewise are out of the mainstream. For this reason you usually discard the first 5-10% of the samples. It is called the *burn-in*. A related problem is the dependence among the samples: two successive samples will in general not be independent, since the second sample is generated by altering the first sample. In this way, these samples are also not representative of the probability distribution, and you therefore typically try to compensate for this by recording samples only at certain intervals.

Another problem is that you may be stuck in certain "areas" of the configurations. Perhaps there is a set of very likely configurations, but in order to reach them from the one you are in, a variable should change to a state that is highly improbable given the remaining configuration (see Exercise 4.43).

Finally, the method relies on an initial starting configuration. Unfortunately, it may be very hard to find such a configuration, and in fact this problem is NP-hard (see Exercise 4.44).

4.9 Loopy Belief Propagation

There is a popular approximate method that is not a version of sampling. It is called *loopy belief propagation* (LBP). LPB has been extremely successful in a setting not directly connected to Bayesian networks, namely in *error-correcting codes*; the so-called *turbo codes*.

LBP is a message passing algorithm similar to the junction tree algorithm in Section 4.4. However, instead of having cliques in a junction tree for passing messages, it uses the nodes in the Bayesian network directly.

The message passing structure consists of one node for each variable in the Bayesian network. A node representing the variable A holds the conditional probability table $P(A \mid \mathrm{pa}(A))$, and it can process potentials over $\mathrm{fa}(A)$ (the variables involved in the table). The neighbors of a node representing A are the neighbors of A in the Bayesian network, and the messages being passed over the links are potentials over the shared variables. We shall stick to the term *separator* for the domains of the potentials being passed over links, though these domains need not separate any variables from others. The structure is illustrated in Figure 4.41.

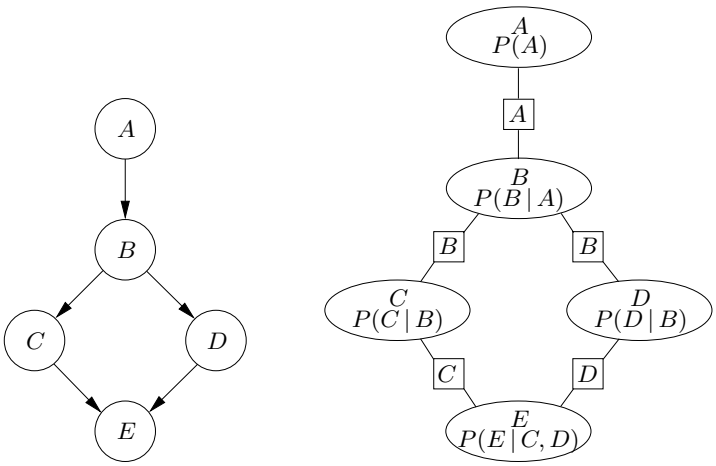
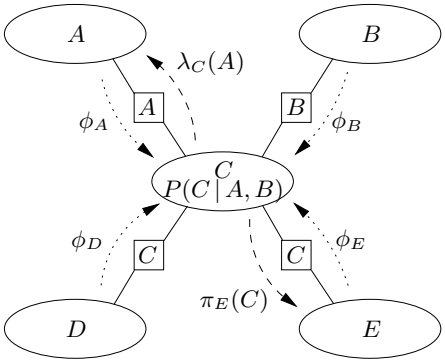


Fig. 4.41. (a) A Bayesian network. (b) The corresponding message-passing structure for LPB. Each node holds the corresponding variable's conditional probability table; the domain of a node is the variable's family. The square box on a link indicates the separator (the domain for the potentials to be passed over that link).

Note that all separators consist of one variable. If B is a child of A then the separator is A.

The processing of messages is similar to the one for junction trees: a message is sent to a neighbor by multiplying the incoming messages from all other

neighbors to the potential it holds and marginalizing the result down to the separator. This is illustrated in Figure 4.42.



$$\lambda_C(A) = \sum_{B,C} P(C \mid A, B)\phi_B\phi_D\phi_E$$

$$\pi_E(C) = \phi_D \sum_{A,B} P(C \mid A, B)\phi_A\phi_B$$

Fig. 4.42. The node C holding $P(C \mid A, B)$ has received all messages (the ϕs). It sends a λ-message to its parent A and a π-message to its child E.

A message from a parent variable to a child variable is called a π-message (because it is in fact a probability distribution), and a message from a child to a parent is called a λ-message (for likelihood).

Since the structure may not be a tree, you cannot use the rule that a node can send to a neighbor when it has received a message from all its other neighbors. In Figure 4.41, only the node A can send a message. All other nodes wait for a message that never comes. Instead, you have a marching regime; at each step all nodes send messages to each neighbor using the messages they have received so far from the other neighbors. After each step, any node A can calculate an estimate of its own probability distribution: take the product of $P(A \mid \mathrm{pa}(A))$ and all incoming messages, marginalize it down to A, and normalize.

Now you let the method march step by step, monitor the development of the probability distributions, and use some stopping criterion. There is no guarantee that the method will converge, nor is there any guarantee that in case of convergence it will converge to the correct posterior distributions. On the other hand, very much experience has been gained, and the method converges to the correct posteriors surprisingly often.

However, sometimes the method is guaranteed to converge correctly, for example, if the network is singly connected (there are no multiple paths in the network). In that case, the junction tree will be exactly the structure for LBP (see Exercise 4.23), and when the method has marched twice the number of links in the network, the messages will be the same as the messages in the junction tree algorithm.

Unfortunately, this result is not of any use. If the Bayesian network is singly connected, the cliques are small, and exact junction tree propagation is no problem. As mentioned above, LBP does very often give good results, and much research is now directed at understanding why and characterizing situations in which you are guaranteed a result within a reasonable margin of tolerance.

4.10 Summary

Exact Belief Updating

Exact belief updating can be performed by message passing in a junction tree representation of the Bayesian network. The junction tree is obtained after triangulating the moral graph of the Bayesian network.

Moral graph: The moral graph of a Bayesian network is obtained by inserting a link between all pairs of variables with a common child, and dropping the direction on all arcs.

Triangulated graph: An undirected graph with a perfect elimination sequence is called a triangulated graph. If a graph is not triangulated, you can insert additional links (determined by, for example, node elimination), making it triangulated.

Node elimination: A node is eliminated by inserting a link between each pair of its noneliminated neighboring nodes.

Perfect elimination sequence: An elimination sequence is perfect if all nodes can be eliminated according to that sequence without inserting a link between a pair of noneliminated variables.

Clique: A complete set is a clique if it is not a subset of another complete set (a maximal complete set).

Join tree: Let $\mathcal{G}$ be the set of cliques from an undirected graph, and let the cliques of $\mathcal{G}$ be organized in a tree T. Then T is a join tree if for any pair of nodes V, W all nodes on the path between V and W contain the intersection $V \cap W$.

Junction tree: Let Φ be a set of potentials with a triangulated domain graph, G. A junction tree for Φ is a join tree for G with the following addition: each potential ϕ in Φ is attached to a clique containing $\mathrm{dom}(\phi)$; each link has the appropriate separator attached; each separator contains two mailboxes, one

for each direction.

Message passing: Let V be a clique with set of potentials Φ_V, and let S be a neighboring separator. Let $S_1, \ldots, S_k$ be the other neighboring separators of V. Assume that each S_i has received a message Ψ_i for V. Then V can pass the message $(\Phi_V \cup \Psi_1 \cup \cdots \cup \Psi_k)^{\downarrow S}$ to S.

Belief updating (calculating marginals): Let the junction tree T represent the Bayesian network BN over the universe U and with evidence e. Assume that each mailbox contains a message.

1. Let V be a clique with set of potentials Φ_V, and let $S_1, \ldots, S_k$ be V's neighboring separators and with V-directed messages $\Psi_1, \ldots, \Psi_k$. Then,
$$P(V, e) = \prod \Phi_V \prod \Psi_1 \cdots \prod \Psi_k.$$
2. Let S be a separator with the sets Ψ_S and Ψ^S in the mailboxes. Then,
$$P(S, e) = \prod \Psi_S \prod \Psi^S.$$

Belief Updating with Bounded Space

If there is not enough space to perform junction tree propagation, you may reduce the space complexity by applying a divide-and-conquer strategy: recursively condition on a variable (or subset of the variables) to be eliminated, solve the new smaller problems, and add up the results. A cache may be introduced to trade space for time.

Approximate Belief Updating

Stochastic simulation: Estimate $P(X \mid e)$ by sampling a large number of random configuration over the variables in the Bayesian network. Throw away the configurations that are inconsistent with e, and let N' be the resulting number of cases. Then
$$P(X \mid e) \approx \frac{N'(X)}{N'}.$$

Likelihood weighting: Estimate $P(X \mid e)$ by sampling a large number of random configurations over the noninstantiated variables in the Bayesian network. Weigh each configuration $(\mathbf{x}, \mathbf{e})$ with
$$w(\mathbf{x}, \mathbf{e}) = \prod_{E \in \mathcal{E}} P(E = e \mid \text{pa}(X) = \pi),$$
where $\mathcal{E}$ are the evidence variables, and π is the configuration of $\text{pa}(X)$ specified by $\mathbf{x}$ and $\mathbf{e}$.

Gibbs sampling: Estimate $P(X \mid e)$ by sampling a large number of random configurations over the noninstantiated variables in the Bayesian network. A sample is generated by starting with a configuration consistent with the evidence, and randomly changing the state of a variable by following the topological order.

Loopy belief updating (LBP): LBP is a message-passing algorithm that works directly on the Bayesian network. Messages are similar to those in junction trees, but in LBP they are passed between the families of variables in the Bayesian network.

4.11 Bibliographical Notes

Loopy belief propagation is rooted in a version of probability updating for singly connected DAGs through message passing presented by Kim and Pearl (1983). In (Pearl, 1986), cutset-conditioning was used to reduce propagation in multiply connected networks to propagation in singly connected networks. Shachter (1986) introduced arc reversal and uses it for a probability updating procedure in the bucket elimination style. Two versions of join tree propagation were presented in the late 1980s. Shafer and Shenoy (1990) proposed the method presented in this book. They did not exploit lazy evaluation but worked with multiplied potentials. Lauritzen and Spiegelhalter (1988) and Jensen *et al.* (1990b) proposed what is now called the Hugin method. It also works with multiplied potentials, but the potentials in the cliques are changed dynamically. This, together with a division operation in the separators, reduced the calculation substantially for join trees with branching higher than three. A detailed study of the similarities and differences of the two methods is reported in (Shafer, 1996). Lazy propagation (Madsen and Jensen, 1999b) dissolves the difference between Shafer-Shenoy and Hugin propagation.

The concepts of triangulated graphs and join trees have been discovered and rediscovered with various names. In (Bertele and Brioschi, 1972), they are used for dynamic programming, and Beeri *et al.* (1983) uses them for database management. A good reference on triangulated graphs is (Golumbic, 1980). The heuristic for triangulating nontriangulated domain graphs given in this chapter is due to Kjærulff (1990), and more can be found in (Cano and Moral, 1995). The problem of inference in dynamic Bayesian networks has been treated in (Boyen and Koller, 1998).

Recursive conditioning was introduced in (Darwiche, 2001). Probabilistic logic sampling was proposed by Henrion (1988), and Fung and Chang (1990) and Shachter and Peot (1990) introduced likelihood-weighted sampling for Bayesian networks. Gibbs sampling was originally introduced for image restoration by Geman and Geman (1984). Gilks *et al.* (1994) have developed a system, BUGS, for Gibbs sampling in Bayesian networks.

4.12 Exercises

Exercise 4.1. *BN* has the potentials in Table 4.3.

	A	
B	y	n
y	0.2	0.6
n	0.8	0.4

$P(B \mid A)$

	B	
C	y	n
y	0.3	0.2
n	0.7	0.8

$P(C \mid B)$

	C	
D	y	n
y	0.9	0.6
n	0.1	0.4

$P(D \mid C)$

Table 4.3. Potentials for Exercise 4.1. $P(A) = (0.2, 0.8)$.

(i) Calculate $P(A \mid D = y)$.
(ii) Calculate $P(C \mid D = y)$.

Exercise 4.2. *BN* has the potentials in Table 4.4.

	A	
B	y	n
y	0.2	0.6
n	0.8	0.4

$P(B \mid A)$

	B	
C	y	n
y	0.1	0.5
n	0.9	0.5

$P(C \mid B)$

	B	
D	y	n
y	0.7	0.4
n	0.3	0.6

$P(D \mid B)$

Table 4.4. Potentials for Exercise 4.2. $P(A) = (0.2, 0.8)$.

(i) Calculate $P(A \mid C = y, D = y)$.
(ii) Calculate $P(A \mid D = y)$.

Exercise 4.3. *BN* has the potentials in Table 4.5.

	A	
B	y	n
y	0.2	0.6
n	0.8	0.4

$P(B \mid A)$

	A	
C	y	n
y	0.1	0.5
n	0.9	0.5

$P(C \mid A)$

	B	
C	y	n
y	(0.3, 0.7)	(0.2, 0.8)
n	(0.9, 0.1)	(0.6, 0.4)

$P(D \mid B, C)$

Table 4.5. Potentials for Exercise 4.3. $P(A) = (0.2, 0.8)$.

(i) Calculate $P(A \mid D = y), P(B \mid D = y), P(C \mid D = y)$.

(ii) Calculate $P(B \mid C = y)$.

Exercise 4.4. Consider the Bayesian network in Figure 4.43. All variables have three states.

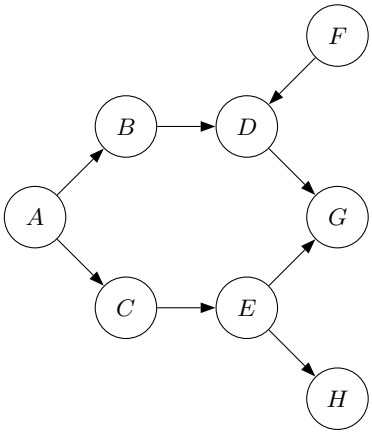


Fig. 4.43. The network for Exercise 4.4.

(i) Calculate the size of the table $P(A, B, C, D, E, F, G = g_1, H = h_1)$.
(ii) In the calculation of $P(A \mid G = g_1, H = h_1)$, the variables have been marginalized in the following order: B, F, D, E, C. Calculate the size of each table produced in the process, and compare the sum with the result of (i).
(iii) Determine an elimination order yielding a sum smaller than the one from (ii).

Exercise 4.5. We have the potentials $\phi_1(A_1, A_2, A_3)$, $\phi_2(A_2, A_3, A_5)$, $\phi_3(A_1, A_3, A_4)$, $\phi_4(A_5, A_6)$ over the universe $\{A_1, A_2, A_3, A_4, A_5, A_6\}$.

(i) Determine the domain graph.
(ii) Eliminate A_3.
(iii) Determine the domain graph for the resulting set of potentials.

Exercise 4.6. We have the potentials $\phi_1(A_1, A_2, A_3)$, $\phi_2(A_2, A_4, A_5)$, $\phi_3(A_4, A_6, A_7)$, $\phi_4(A_1, A_6, A_8)$ over the universe $\{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$.

(i) Determine the domain graph.
(ii) Eliminate A_1.
(iii) Determine the domain graph for the resulting set of potentials.

Exercise 4.7. Write a short algorithm that takes as input a Bayesian network over nodes $X_1, \ldots, X_n$ and an elimination sequence for all nodes but X_i, and which outputs the maximum table size that would be used during computation of $P(X_i)$ using this elimination sequence.

Exercise 4.8. Consider the Bayesian network given in Figure 4.44. What would the elimination trees (such as those in Figures 4.2 to 4.7) look like for the two elimination orders C, F, G, B, E, D and F, E, G, D, C, B?

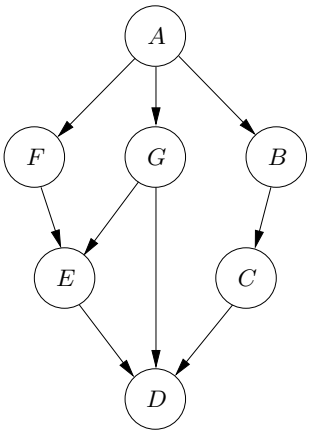


Fig. 4.44. A Bayesian network.

Exercise 4.9. Prove Proposition 4.1.

Exercise 4.10. What is $(\prod \phi)^{\downarrow A}$ for the Bayesian network in Figure 4.44?

Exercise 4.11. What are the domains encoded by the domain graph in Figure 4.45? Give an example of an elimination sequence ending with C. What do the intermediate domain graphs look like as you apply the elimination sequence? Is the sequence perfect?

Exercise 4.12. Consider the domain graph for the potentials in Exercise 4.5. Determine a perfect elimination sequence ending with A_1.

Exercise 4.13. Consider the domain graph for the potentials in Exercise 4.6. Does the graph have a perfect elimination sequence?

Exercise 4.14. Consider the Bayesian network in Figure 4.43.

(i) Determine the domain graph.
(ii) Does the domain graph have a perfect elimination sequence?

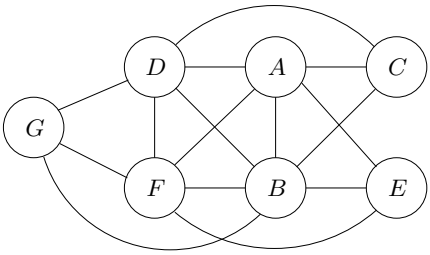


Fig. 4.45. A domain graph.

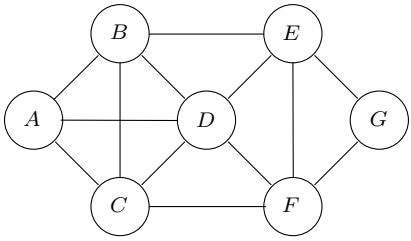


Fig. 4.46. The graph for Exercise 4.15.

Exercise 4.15. Consider the graph in Figure 4.46.

(i) Determine the simplicial nodes.
(ii) Is the graph triangulated?

Exercise 4.16. Consider the graph in Figure 4.47.

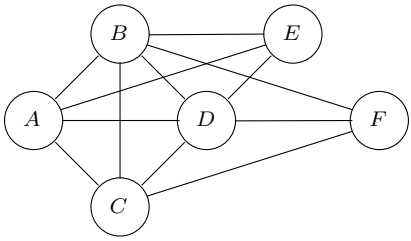


Fig. 4.47. The graph for Exercise 4.16.

(i) Determine the simplicial nodes.
(ii) Is the graph triangulated?

Exercise 4.17. Definition Let G be an undirected graph with node set U. A *path* in G is a sequence $A_1, \ldots, A_n$ of distinct nodes; where A_i and A_{i+1} are neighbors. A *cycle* is a path except $A_1 = A_n$, and all other nodes are distinct. A *chord* in a cycle $A_1, \ldots, A_n$ is a link between two nodes A_i and A_j that are not neighbors on the path. The graph G is *chord-saturated* if any cycle of length > 3 has a chord.

(i) Prove that any triangulated graph is chord-saturated. (Hint: Use induction and the fact that any cycle through a simplicial node must have a chord.)
(ii) Prove the following decomposition lemma. Let G be a incomplete chord-saturated graph with at least three nodes and with node set U. Then there is a complete subset S of U such that $G \setminus S$ is disconnected. (Hint: Let A and B be two nonadjacent nodes, and let S be a minimal set of nodes such that any path connecting A and B contains a node from S. Use chord saturation and minimality of S to prove that S is complete.)
(iii) Prove that any chord-saturated graph is triangulated. (Hint: Use (ii) to prove that any incomplete chord-saturated graph with at least two nodes has at least two simplicial nodes.)

Exercise 4.18. Prove that the moral graph of the graph in Figure 4.48 is triangulated. Give an example of a join tree for the graph.

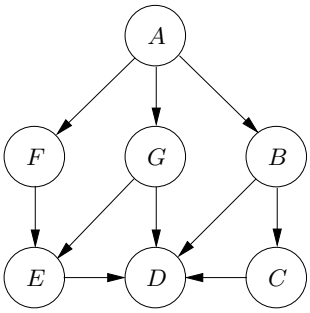


Fig. 4.48. A Bayesian network.

Exercise 4.19. Consider the domain graph from Exercise 4.5.

(i) Determine the cliques.
(ii) Construct a join tree for the graph.

Exercise 4.20. Consider the graph in Figure 4.47.

(i) Determine the cliques.

(ii) Construct a join tree for the graph.

Exercise 4.21. Consider the Bayesian network in Figure 4.49. Construct a join tree.

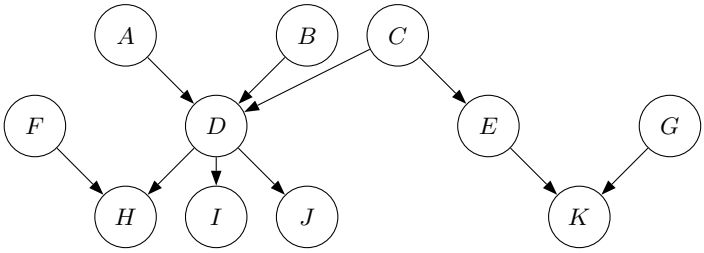


Fig. 4.49. The Bayesian network for Exercise 4.21.

Exercise 4.22. Let $\mathcal{A}$ and $\mathcal{B}$ be any two adjacent nodes in a join tree for a Bayesian network M with separator $\mathcal{S} = \mathcal{A} \cap \mathcal{B}$. Furthermore, let $\mathcal{U}_{\mathcal{A}}$ be the variables in the nodes found in the part of the join tree on $\mathcal{A}$'s side of the link, and $\mathcal{U}_{\mathcal{B}}$ those found in nodes on $\mathcal{B}$'s side of the link. Prove that for any two nodes $A \in \mathcal{U}_{\mathcal{A}} \setminus \mathcal{S}$ and $B \in \mathcal{U}_{\mathcal{B}} \setminus \mathcal{S}$, we have that A and B are d-separated by $\mathcal{S}$.

Exercise 4.23. A directed acyclic graph is *singly connected* if the graph you get by dropping the directions of the links is a tree (the graph in Figure 4.49 is singly connected).

(i) Prove that the moral graph of a singly connected graph is triangulated. (Hint: If you successively eliminate a node with exactly one parent and no children or with no parents and exactly one child, then the result is a moral graph for a singly connected graph.)
(ii) Prove that the separators in a join tree for a singly connected graph consist of exactly one node. (Hint: If the neighbors A and B share the neighbors C and D, then C and D are neighbors.)

Exercise 4.24. Consider the Bayesian network in Exercise 4.21.

Indicate the potentials to communicate in a full lazy propagation with evidence $F = f$, $I = i$, $E = e$.

Exercise 4.25. Expand the join tree in Figure 4.16 to a junction tree, and add the potentials defined by the domain graph in Figure 4.14 to suitable cliques. Which messages are sent if evidence is collected to node CG?

Exercise 4.26. Consider the Bayesian network in Figure 4.50.

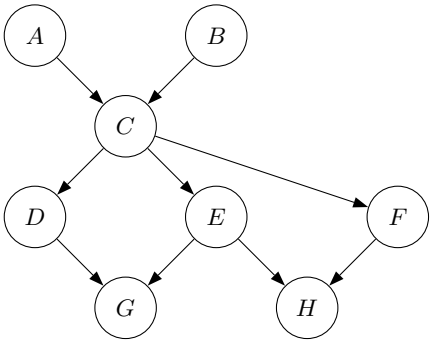


Fig. 4.50. The Bayesian network for Exercise 4.26.

(i) Construct a junction tree.
(ii) Indicate the potentials to communicate in a full lazy propagation without evidence.
(iii) Indicate the potentials to communicate with evidence $D = d$ and $H = h$.

Exercise 4.27. Prove Proposition 4.5. (Hint: Assume a deadlock (no triggered nodes).)

Exercise 4.28. Show that any asynchronous full order of message passing corresponds to a CollectEvidence(R) followed by a DistributeEvidence(R) for some node R. (Hint: Look at the first node that receives all its messages.)

Exercise 4.29. Triangulate the domain graph from Exercise 4.6.

Exercise 4.30.

(i) Construct a junction tree for the Bayesian network in Figure 4.51 by using the elimination order F, J, B, A, I, K, E.
(ii) The numbers inside the nodes indicate the number of states. Use the heuristics from Section 4.6.1 to construct a junction tree.

Exercise 4.31. What is the moral graph of the Bayesian network in Figure 4.44? Assuming that each node has 10 states, use the heuristics following Definition 4.8 to triangulate the graph. Would the result be the same if each node had 2 states instead?

Exercise 4.32. Consider the Bayesian network in Figure 4.29, and let the number of states be as listed in Section 4.6.1. Find a better triangulation than the one obtained by using the heuristics from Section 4.6.1.

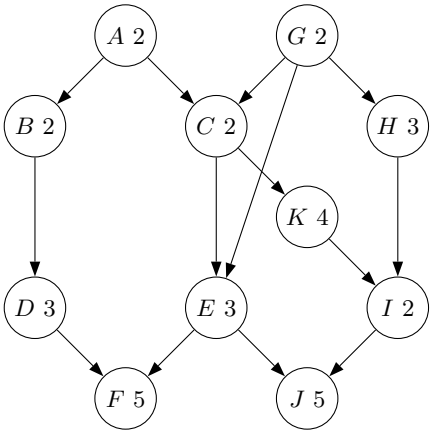


Fig. 4.51. The Bayesian network for Exercise 4.30.

Exercise 4.33. (Conditioning) Propagation methods for DAGs without multiple paths have existed for a long time. A propagation method for multiply connected DAGs consists in reducing a DAG to a set of singly connected DAGs.

(i) Consider the DAG (a) in Figure 4.52 with $P(A), P(B \mid A), P(C \mid A)$, and $P(D \mid B, C)$ given. Assume that $A = a$. Show that the DAG is reduced to the DAG (b) with $P(B, a), P(C, a)$, and $P(D \mid B, C)$ given. (Hint: Use the chain rule.. Calculate $P(B, a)$ and $P(C, a)$.

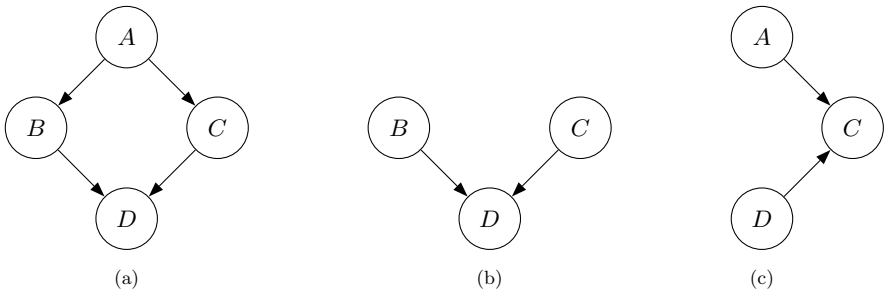


Fig. 4.52. Figures for Exercise 4.33(i)–(v).

(ii) Show that $P(D, a) = \sum_{B,C} P(D \mid B, C)P(B, a)P(C, a)$.

(iii) Assume that for all states a of A we have a reduced DAG as in (i). Let evidence e be entered and propagated in all the reduced DAGs, yielding $P(B, a, e), P(C, a, e), P(D, a, e)$ for all a. Calculate $P(B, e)$ and $P(A, e)$. This procedure is called *conditioning on A*.

(iv) Reduce the DAG by conditioning on B. Show that the tables are $P(A, b)$, $P(C \mid A)$, and $P(D \mid C, b)$.
(v) Show that conditioning on D does not result in a singly connected DAG. Conditioning over several variables can be performed stepwise.
(vi) Determine a minimal set of conditioning variables for the DAG in Figure 4.53 to reduce it to singly connected DAGs.

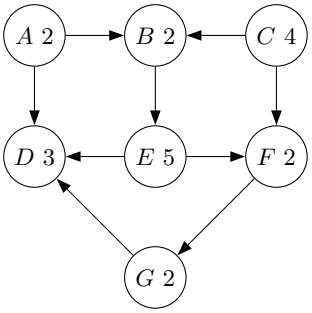


Fig. 4.53. Figure for Exercise 4.33 (vi)–(vii).

(vii) The numbers attached to the variables indicate the number of states. Determine a conditioning resulting in a minimal number of singly connected DAGs.

Exercise 4.34. Let $\mathcal{C}$ be the set of cliques from a triangulated graph. A *pre-$\mathcal{J}$-tree* is a tree over $\mathcal{C}$ with separators $S = V \cap W$ for adjacent cliques V, W. The *weight* of a pre-$\mathcal{J}$-tree is the sum of the number of variables in the separators.

(i) Prove that a join tree is a pre-$\mathcal{J}$-tree of maximal weight.
(ii) Prove that any pre-$\mathcal{J}$-tree of maximal weight is a join tree.

Exercise 4.35. (i) Consider the graph in Figure 4.35. Determine a triangulation such that no clique contains more than four nodes.
(ii) Expand the model in Figure 4.34 to six time slices. Can this model be triangulated such that no clique contains more than four nodes?

Exercise 4.36. Consider the Bayesian network in Figure 4.54, where each variable is binary, with probabilities defined as $P(A = a_1) = 0.1$, $P(B = b_1 \mid a_1) = 0.1$, $P(B = b_1 \mid a_2) = 0.9$, $P(C = c_1 \mid b_1) = 0.1$, $P(C = c_1 \mid b_2) = 0.9$, $P(D = d_1 \mid c_1) = 0.1$, and $P(D = d_1 \mid c_2) = 0.9$. Using recursive conditioning, calculate $P(a_1 \mid d_1)$.

Exercise 4.37. Construct two time slices of the model in Figure 3.52. Using recursive conditioning, what would a computation tree for calculating $P(C_2)$ look like?

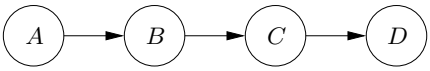


Fig. 4.54. A simple Bayesian network.

Exercise 4.38. Show that the worst case complexity of Algorithm 4.2 is $O(n \cdot \exp(wn))$, and that the complexity for a balanced tree is $O(n^{w+1})$.

Exercise 4.39. Calculate the marginals from the sample in Table 4.2 and compare the result with the exact marginals.

Exercise 4.40. From the configurations in Table 4.2, estimate the following probability distributions: $P(A)$, $P(A \mid D = n)$, and $P(C, D \mid B = y, E = n)$.

Exercise 4.41. Does your software tool allow for sampling from a Bayesian network model? Which kind of sampling technique is used?

Exercise 4.42. Using the sequence of random numbers in Table 4.41 generate as many full samples as you can for the Bayesian network model given in Figure 4.46, with conditional probabilities as defined in Table 4.1 and evidence $B = n$, using first probabilistic logic sampling, then likelihood weighting, then Gibbs sampling using sampling sequence A, C, D, E, and finally Gibbs sampling using sampling sequence A, D, E, C.

1 0.80	5 0.33	9 0.55	13 0.14
2 0.19	6 0.08	10 0.71	14 0.42
3 0.85	7 0.52	11 0.06	15 0.32
4 0.28	8 0.65	12 0.78	16 0.11

Table 4.6. A sequence of random numbers in the interval $[0, 1]$.

Exercise 4.43. The binary variables A and B are parents of the binary variable C. We have $P(A) = P(B) = (0.5, 0.5)$, and the conditional probability table is an exclusive OR table ($C = y$ if and only if exactly one of A and B is in the state y). Show that Gibbs sampling on this structure will give either $P(C = y) = 1$ or $P(C = n) = 1$.

Exercise 4.44. Given a Bayesian network over U with evidence e entered, show that it is NP-hard to find a configuration U^* such that $P(U^*, e) > 0$. (Hint: Look at Exercise 3.27.)