variables, and then a variable X whose value, given any assignment of values q_1^1, q_1^0 to the Q_i's, was the associated truth value of the formula. We now show that, given such an approximation algorithm, we can decide whether the formula is satisfiable. We begin by computing $P(Q_1 \mid x^1)$. We pick the value v_1 for Q_1 that is most likely given x^1, and we instantiate it to this value. That is, we generate a network $\mathcal{B}_2$ that does not contain Q_1, and that represents the distribution $\mathcal{B}$ conditioned on $Q_1 = v_1$. We repeat this process for $Q_2, \ldots, Q_n$. This results in some assignment $v_1, \ldots, v_n$ to the Q_i's. We now prove that this is a satisfying assignment if and only if the original formula ϕ was satisfiable.

We begin with the easy case. If ϕ is not satisfiable, then $v_1, \ldots, v_n$ can hardly be a satisfying assignment for it. Now, assume that ϕ is satisfiable. We show that it also has a satisfying assignment with $Q_1 = v_1$. If ϕ is satisfiable with both $Q_1 = q_1^1$ and $Q_1 = q_1^0$, then this is obvious. Assume, however, that ϕ is satisfiable, but not when $Q_1 = v$. Then necessarily, we will have that $P(Q_1 = v \mid x^1)$ is 0, and the probability of the complementary event is 1. If we have an approximation ρ whose error is guaranteed to be $< 1/2$, then choosing the v that maximizes this probability is guaranteed to pick the v whose probability is 1. Thus, in either case the formula has a satisfying assignment where $Q_1 = v$.

We can continue in this fashion, proving by induction on k that ϕ has a satisfying assignment with $Q_1 = v_1, \ldots, Q_k = v_k$. In the case where ϕ is satisfiable, this process will terminate with a satisfying assignment. In the case where ϕ is not, it clearly will not terminate with a satisfying assignment. We can determine which is the case simply by checking whether the resulting assignment satisfies ϕ. This gives us a polynomial time process for deciding satisfiability. ■

Because $\epsilon = 1/2$ corresponds to random guessing, this result is quite discouraging. It tells us that, in the case where we have evidence, approximate inference is no easier than exact inference, in the worst case.

9.2 Variable Elimination: The Basic Ideas

We begin our discussion of inference by discussing the principles underlying exact inference in graphical models. As we show, the same graphical structure that allows a compact representation of complex distributions also help support inference. In particular, we can use dynamic programming techniques (as discussed in appendix A.3.3) to perform inference even for certain large and complex networks in a very reasonable time. We now provide the intuition underlying these algorithms, an intuition that is presented more formally in the remainder of this chapter.

We begin by considering the inference task in a very simple network $A \rightarrow B \rightarrow C \rightarrow D$. We first provide a phased computation, which uses results from the previous phase for the computation in the next phase. We then reformulate this process in terms of a global computation on the joint distribution.

Assume that our first goal is to compute the probability $P(B)$, that is, the distribution over values b of B. Basic probabilistic reasoning (with no assumptions) tells us that

$$P(B) = \sum_a P(a)P(B \mid a). \tag{9.4}$$

Fortunately, we have all the required numbers in our Bayesian network representation: each number $P(a)$ is in the CPD for A, and each number $P(b \mid a)$ is in the CPD for B. Note that if A has k values and B has m values, the number of basic arithmetic operations required is $O(k \times m)$: to compute $P(b)$, we must multiply $P(b \mid a)$ with $P(a)$ for each of the k values of A, and then add them up, that is, k multiplications and $k-1$ additions; this process must be repeated for each of the m values b.

Now, assume we want to compute $P(C)$. Using the same analysis, we have that

$$P(C) = \sum_b P(b)P(C \mid b). \tag{9.5}$$

Again, the conditional probabilities $P(c \mid b)$ are known: they constitute the CPD for C. The probability of B is not specified as part of the network parameters, but equation (9.4) shows us how it can be computed. Thus, we can compute $P(C)$. We can continue the process in an analogous way, in order to compute $P(D)$.

Note that the structure of the network, and its effect on the parameterization of the CPDs, is critical for our ability to perform this computation as described. Specifically, assume that A had been a parent of C. In this case, the CPD for C would have included A, and our computation of $P(B)$ would not have sufficed for equation (9.5).

Also note that this algorithm does not compute single values, but rather sets of values at a time. In particular equation (9.4) computes an entire distribution over all of the possible values of B. All of these are then used in equation (9.5) to compute $P(C)$. This property turns out to be critical for the performance of the general algorithm.

Let us analyze the complexity of this process on a general chain. Assume that we have a chain with n variables $X_1 \to \ldots \to X_n$, where each variable in the chain has k values. As described, the algorithm would compute $P(X_{i+1})$ from $P(X_i)$, for $i = 1, \ldots, n-1$. Each such step would consist of the following computation:

$$P(X_{i+1}) = \sum_{x_i} P(X_{i+1} \mid x_i)P(x_i),$$

where $P(X_i)$ is computed in the previous step. The cost of each such step is $O(k^2)$: The distribution over X_i has k values, and the CPD $P(X_{i+1} \mid X_i)$ has k^2 values; we need to multiply $P(x_i)$, for each value x_i, with each CPD entry $P(x_{i+1} \mid x_i)$ (k^2 multiplications), and then, for each value x_{i+1}, sum up the corresponding entries ($k \times (k-1)$ additions). We need to perform this process for every variable $X_2, \ldots, X_n$; hence, the total cost is $O(nk^2)$.

By comparison, consider the process of generating the entire joint and summing it out, which requires that we generate k^n probabilities for the different events $x_1, \ldots, x_n$. Hence, we have at least one example where, despite the exponential size of the joint distribution, we can do inference in linear time.

Using this process, we have managed to do inference over the joint distribution without ever generating it explicitly. What is the basic insight that allows us to avoid the exhaustive enumeration? Let us reexamine this process in terms of the joint $P(A, B, C, D)$. By the chain rule for Bayesian networks, the joint decomposes as

$$P(A)P(B \mid A)P(C \mid B)P(D \mid C)$$

To compute $P(D)$, we need to sum together all of the entries where $D = d^1$, and to (separately) sum together all of the entries where $D = d^2$. The exact computation that needs to be

$$\begin{array}{rllll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2)
\end{array}$$

$$\begin{array}{rllll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.2 Computing $P(D)$ by summing over the joint distribution for a chain $A \to B \to C \to D$; all of the variables are binary valued.

performed, for binary-valued variables A, B, C, D, is shown in figure 9.2.[1]

Examining this summation, we see that it has a lot of structure. For example, the third and fourth terms in the first two entries are both $P(c^1 \mid b^1)P(d^1 \mid c^1)$. We can therefore modify the computation to first compute

$$P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)$$

and only then multiply by the common term. The same structure is repeated throughout the table. If we perform the same transformation, we get a new expression, as shown in figure 9.3.

We now observe that certain terms are repeated several times in this expression. Specifically, $P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)$ and $P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)$ are each repeated four times. Thus, it seems clear that we can gain significant computational savings by computing them once and then storing them. There are two such expressions, one for each value of B. Thus, we define a function $\tau_1 : Val(B) \mapsto \mathbb{R}$, where $\tau_1(b^1)$ is the first of these two expressions, and $\tau_1(b^2)$ is the second. Note that $\tau_1(B)$ corresponds exactly to $P(B)$.

The resulting expression, assuming $\tau_1(B)$ has been computed, is shown in figure 9.4. Examining this new expression, we see that we once again can reverse the order of a sum and a product, resulting in the expression of figure 9.5. And, once again, we notice some shared expressions, that are better computed once and used multiple times. We define $\tau_2 : Val(C) \mapsto \mathbb{R}$.

$$\begin{aligned}
\tau_2(c^1) &= \tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2) \\
\tau_2(c^2) &= \tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)
\end{aligned}$$

1. When D is binary-valued, we can get away with doing only the first of these computations. However, this trick does not carry over to the case of variables with more than two values or to the case where we have evidence. Therefore, our example will show the computation in its generality.

$$\begin{array}{rlll}
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
\\
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.3 The first transformation on the sum of figure 9.2

$$\begin{array}{rlll}
 & \tau_1(b^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
\\
 & \tau_1(b^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.4 The second transformation on the sum of figure 9.2

$$\begin{array}{rll}
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^1 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^1 \mid c^2) \\
\\
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^2 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.5 The third transformation on the sum of figure 9.2

$$\begin{array}{rll}
 & \tau_2(c^1) & P(d^1 \mid c^1) \\
+ & \tau_2(c^2) & P(d^1 \mid c^2) \\
\\
 & \tau_2(c^1) & P(d^2 \mid c^1) \\
+ & \tau_2(c^2) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.6 The fourth transformation on the sum of figure 9.2

The final expression is shown in figure 9.6.

Summarizing, we begin by computing $\tau_1(B)$, which requires four multiplications and two additions. Using it, we can compute $\tau_2(C)$, which also requires four multiplications and two additions. Finally, we can compute $P(D)$, again, at the same cost. The total number of operations is therefore 18. By comparison, generating the joint distribution requires $16 \cdot 3 = 48$

multiplications (three for each of the 16 entries in the joint), and 14 additions (7 for each of $P(d^1)$ and $P(d^2)$).

Written somewhat more compactly, the transformation we have performed takes the following steps: We want to compute

$$P(D) = \sum_C \sum_B \sum_A P(A)P(B \mid A)P(C \mid B)P(D \mid C).$$

We push in the first summation, resulting in

$$\sum_C P(D \mid C) \sum_B P(C \mid B) \sum_A P(A)P(B \mid A).$$

We compute the product $\psi_1(A, B) = P(A)P(B \mid A)$ and then sum out A to obtain the function $\tau_1(B) = \sum_A \psi_1(A, B)$. Specifically, for each value b, we compute $\tau_1(b) = \sum_A \psi_1(A, b) = \sum_A P(A)P(b \mid A)$. We then continue by computing:

$$\begin{aligned} \psi_2(B, C) &= \tau_1(B)P(C \mid B) \\ \tau_2(C) &= \sum_B \psi_2(B, C). \end{aligned}$$

This computation results in a new vector $\tau_2(C)$, which we then proceed to use in the final phase of computing $P(D)$.

dynamic programming

This procedure is performing *dynamic programming* (see appendix A.3.3); doing this summation the naive way would have us compute every $P(b) = \sum_A P(A)P(b \mid A)$ many times, once for every value of C and D. In general, in a chain of length n, this internal summation would be computed exponentially many times. Dynamic programming "inverts" the order of computation — performing it inside out instead of outside in. Specifically, we perform the innermost summation first, computing once and for all the values in $\tau_1(B)$; that allows us to compute $\tau_2(C)$ once and for all, and so on.

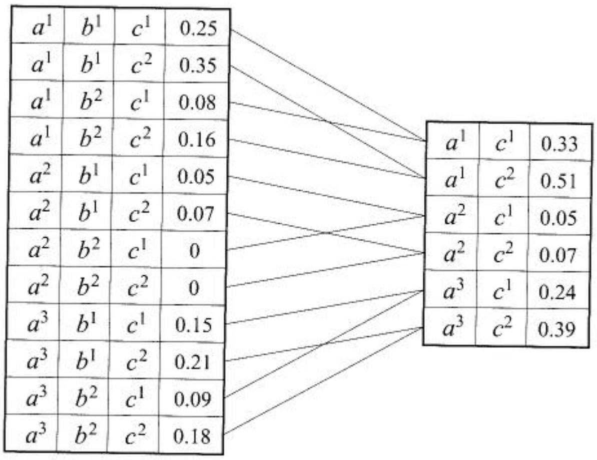
☞ **To summarize, the two ideas that help us address the exponential blowup of the joint distribution are:**

- **Because of the structure of the Bayesian network, some subexpressions in the joint depend only on a small number of variables.**
- **By computing these expressions once and caching the results, we can avoid generating them exponentially many times.**

9.3 Variable Elimination

factor

To formalize the algorithm demonstrated in the previous section, we need to introduce some basic concepts. In chapter 4, we introduced the notion of a *factor* ϕ over a scope *Scope*$[\phi] = \boldsymbol{X}$, which is a function $\phi : Val(\boldsymbol{X}) \mapsto \mathbb{R}$. The main steps in the algorithm described here can be viewed as a manipulation of factors. Importantly, by using the factor-based view, we can define the algorithm in a general form that applies equally to Bayesian networks and Markov networks.



a^1	b^1	c^1	0.25
a^1	b^1	c^2	0.35
a^1	b^2	c^1	0.08
a^1	b^2	c^2	0.16
a^2	b^1	c^1	0.05
a^2	b^1	c^2	0.07
a^2	b^2	c^1	0
a^2	b^2	c^2	0
a^3	b^1	c^1	0.15
a^3	b^1	c^2	0.21
a^3	b^2	c^1	0.09
a^3	b^2	c^2	0.18

a^1	c^1	0.33
a^1	c^2	0.51
a^2	c^1	0.05
a^2	c^2	0.07
a^3	c^1	0.24
a^3	c^2	0.39

Figure 9.7 Example of factor marginalization: summing out B.

9.3.1 Basic Elimination

9.3.1.1 Factor Marginalization

The key operation that we are performing when computing the probability of some subset of variables is that of marginalizing out variables from a distribution. That is, we have a distribution over a set of variables $\mathcal{X}$, and we want to compute the marginal of that distribution over some subset $\boldsymbol{X}$. We can view this computation as an operation on a factor:

Definition 9.3 factor marginalization

Let $\boldsymbol{X}$ be a set of variables, and $Y \notin \boldsymbol{X}$ a variable. Let $\phi(\boldsymbol{X}, Y)$ be a factor. We define the factor marginalization *of Y in ϕ, denoted $\sum_Y \phi$, to be a factor ψ over $\boldsymbol{X}$ such that:*

$$\psi(\boldsymbol{X}) = \sum_Y \phi(\boldsymbol{X}, Y).$$

This operation is also called summing out *of Y in ψ.* ■

The key point in this definition is that we only sum up entries in the table where the values of $\boldsymbol{X}$ match up. Figure 9.7 illustrates this process.

The process of marginalizing a joint distribution $P(\boldsymbol{X}, \boldsymbol{Y})$ onto $\boldsymbol{X}$ in a Bayesian network is simply summing out the variables $\boldsymbol{Y}$ in the factor corresponding to P. If we sum out all variables, we get a factor consisting of a single number whose value is 1. If we sum out all of the variables in the unnormalized distribution $\tilde{P}_\Phi$ defined by the product of factors in a Markov network, we get the partition function.

A key observation used in performing inference in graphical models is that the operations of factor product and summation behave precisely as do product and summation over numbers. Specifically, both operations are commutative, so that $\phi_1 \cdot \phi_2 = \phi_2 \cdot \phi_1$ and $\sum_X \sum_Y \phi = \sum_Y \sum_X \phi$. Products are also associative, so that $(\phi_1 \cdot \phi_2) \cdot \phi_3 = \phi_1 \cdot (\phi_2 \cdot \phi_3)$. Most importantly,

Algorithm 9.1 Sum-product variable elimination algorithm

```
    Procedure Sum-Product-VE (
       $\Phi$,   // Set of factors
       $\boldsymbol{Z}$,   // Set of variables to be eliminated
       $\prec$   // Ordering on $\boldsymbol{Z}$
    )
1      Let $Z_1, \ldots, Z_k$ be an ordering of $\boldsymbol{Z}$ such that
2        $Z_i \prec Z_j$ if and only if $i < j$
3      for $i = 1, \ldots, k$
4        $\Phi \leftarrow$ Sum-Product-Eliminate-Var($\Phi, Z_i$)
5      $\phi^* \leftarrow \prod_{\phi \in \Phi} \phi$
6      return $\phi^*$

    Procedure Sum-Product-Eliminate-Var (
       $\Phi$,   // Set of factors
       $Z$   // Variable to be eliminated
    )
1      $\Phi' \leftarrow \{\phi \in \Phi : Z \in Scope[\phi]\}$
2      $\Phi'' \leftarrow \Phi - \Phi'$
3      $\psi \leftarrow \prod_{\phi \in \Phi'} \phi$
4      $\tau \leftarrow \sum_Z \psi$
5      return $\Phi'' \cup \{\tau\}$
```

we have a simple rule allowing us to exchange summation and product: If $X \notin Scope[\phi_1]$, then

$$\sum_X (\phi_1 \cdot \phi_2) = \phi_1 \cdot \sum_X \phi_2. \tag{9.6}$$

9.3.1.2 The Variable Elimination Algorithm

The key to both of our examples in the last section is the application of equation (9.6). Specifically, in our chain example of section 9.2, we can write:

$$P(A, B, C, D) = \phi_A \cdot \phi_B \cdot \phi_C \cdot \phi_D.$$

On the other hand, the marginal distribution over D is

$$P(D) = \sum_C \sum_B \sum_A P(A, B, C, D).$$

Applying equation (9.6), we can now conclude:

$$\begin{aligned} P(D) &= \sum_C \sum_B \sum_A \phi_A \cdot \phi_B \cdot \phi_C \cdot \phi_D \\ &= \sum_C \sum_B \phi_C \cdot \phi_D \cdot \left(\sum_A \phi_A \cdot \phi_B \right) \\ &= \sum_C \phi_D \cdot \left(\sum_B \phi_C \cdot \left(\sum_A \phi_A \cdot \phi_B \right) \right), \end{aligned}$$

where the different transformations are justified by the limited scope of the CPD factors; for example, the second equality is justified by the fact that the scope of ϕ_C and ϕ_D does not contain A. In general, any marginal probability computation involves taking the product of all the CPDs, and doing a summation on all the variables except the query variables. We can do these steps in any order we want, as long as we only do a summation on a variable X *after* multiplying in all of the factors that involve X.

In general, we can view the task at hand as that of computing the value of an expression of the form:

$$\sum_{\boldsymbol{Z}} \prod_{\phi \in \Phi} \phi.$$

sum-product

We call this task the *sum-product* inference task. The key insight that allows the effective computation of this expression is the fact that the scope of the factors is limited, allowing us to "push in" some of the summations, performing them over the product of only a subset of factors. One simple instantiation of this algorithm is a procedure called *sum-product variable elimination* (VE), shown in algorithm 9.1. The basic idea in the algorithm is that we sum out variables one at a time. When we sum out any variable, we multiply all the factors that mention that variable, generating a product factor. Now, we sum out the variable from this combined factor, generating a new factor that we enter into our set of factors to be dealt with.

variable elimination

Based on equation (9.6), the following result follows easily:

Theorem 9.5

Let $\boldsymbol{X}$ be some set of variables, and let Φ be a set of factors such that for each $\phi \in \Phi$, Scope$[\phi] \subseteq \boldsymbol{X}$. Let $\boldsymbol{Y} \subset \boldsymbol{X}$ be a set of query variables, and let $\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{Y}$. Then for any ordering $\prec$ over $\boldsymbol{Z}$, Sum-Product-VE($\Phi, \boldsymbol{Z}, \prec$) *returns a factor $\phi^*(\boldsymbol{Y})$ such that*

$$\phi^*(\boldsymbol{Y}) = \sum_{\boldsymbol{Z}} \prod_{\phi \in \Phi} \phi.$$

We can apply this algorithm to the task of computing the probability distribution $P_{\mathcal{B}}(\boldsymbol{Y})$ for a Bayesian network $\mathcal{B}$. We simply instantiate Φ to consist of all of the CPDs:

$$\Phi = \{\phi_{X_i}\}_{i=1}^n$$

where $\phi_{X_i} = P(X_i \mid \mathrm{Pa}_{X_i})$. We then apply the variable elimination algorithm to the set $\{Z_1, \ldots, Z_m\} = \mathcal{X} - \boldsymbol{Y}$ (that is, we eliminate all the nonquery variables).

We can also apply precisely the same algorithm to the task of computing conditional probabilities in a Markov network. We simply initialize the factors to be the clique potentials and

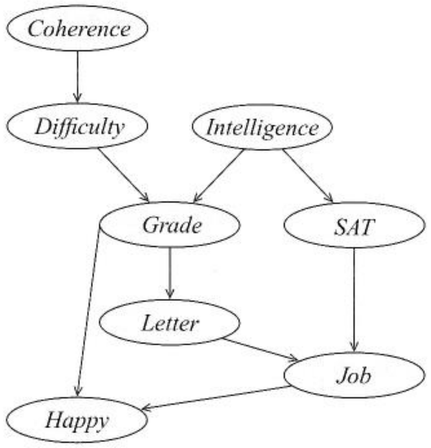


Figure 9.8 **The** Extended-Student **Bayesian network**

run the elimination algorithm. As for Bayesian networks, we then apply the variable elimination algorithm to the set $\boldsymbol{Z} = \mathcal{X} - \boldsymbol{Y}$. The procedure returns an *unnormalized* factor over the query variables $\boldsymbol{Y}$. The distribution over $\boldsymbol{Y}$ can be obtained by normalizing the factor; the partition function is simply the normalizing constant.

Example 9.1 *Let us demonstrate the procedure on a nontrivial example. Consider the network demonstrated in figure 9.8, which is an extension of our* Student *network. The chain rule for this network asserts that*

$$\begin{aligned} P(C,D,I,G,S,L,J,H) &= P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\ &\quad P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\ &= \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\ &\quad \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J). \end{aligned}$$

We will now apply the VE algorithm to compute $P(J)$. We will use the elimination ordering: C, D, I, H, G, S, L:

1. *Eliminating C: We compute the factors*

$$\begin{aligned} \psi_1(C,D) &= \phi_C(C) \cdot \phi_D(D,C) \\ \tau_1(D) &= \sum_C \psi_1. \end{aligned}$$

2. *Eliminating D: Note that we have already eliminated one of the original factors that involve D — $\phi_D(D,C) = P(D \mid C)$. On the other hand, we introduced the factor $\tau_1(D)$ that involves*

D. Hence, we now compute:

$$\begin{aligned}\psi_2(G,I,D) &= \phi_G(G,I,D)\cdot\tau_1(D)\\ \tau_2(G,I) &= \sum_D \psi_2(G,I,D).\end{aligned}$$

3. *Eliminating I: We compute the factors*

$$\begin{aligned}\psi_3(G,I,S) &= \phi_I(I)\cdot\phi_S(S,I)\cdot\tau_2(G,I)\\ \tau_3(G,S) &= \sum_I \psi_3(G,I,S).\end{aligned}$$

4. *Eliminating H: We compute the factors*

$$\begin{aligned}\psi_4(G,J,H) &= \phi_H(H,G,J)\\ \tau_4(G,J) &= \sum_H \psi_4(G,J,H).\end{aligned}$$

Note that $\tau_4 \equiv 1$ (all of its entries are exactly 1): we are simply computing $\sum_H P(H \mid G,J)$, which is a probability distribution for every G, J, and hence sums to 1. A naive execution of this algorithm will end up generating this factor, which has no value. Generating it has no impact on the final answer, but it does complicate the algorithm. In particular, the existence of this factor complicates our computation in the next step.

5. *Eliminating G: We compute the factors*

$$\begin{aligned}\psi_5(G,J,L,S) &= \tau_4(G,J)\cdot\tau_3(G,S)\cdot\phi_L(L,G)\\ \tau_5(J,L,S) &= \sum_G \psi_5(G,J,L,S).\end{aligned}$$

Note that, without the factor $\tau_4(G,J)$, the results of this step would not have involved J.

6. *Eliminating S: We compute the factors*

$$\begin{aligned}\psi_6(J,L,S) &= \tau_5(J,L,S)\cdot\phi_J(J,L,S)\\ \tau_6(J,L) &= \sum_S \psi_6(J,L,S).\end{aligned}$$

7. *Eliminating L: We compute the factors*

$$\begin{aligned}\psi_7(J,L) &= \tau_6(J,L)\\ \tau_7(J) &= \sum_L \psi_7(J,L).\end{aligned}$$

We summarize these steps in table 9.1.

Note that we can use any elimination ordering. For example, consider eliminating variables in the order G, I, S, L, H, C, D. We would then get the behavior of table 9.2. The result, as before, is precisely $P(J)$. However, note that this elimination ordering introduces factors with much larger scope. We return to this point later on. ■

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D, C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G, I, D)$, $\tau_1(D)$	G, I, D	$\tau_2(G, I)$
3	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_2(G, I)$	G, S, I	$\tau_3(G, S)$
4	H	$\phi_H(H, G, J)$	H, G, J	$\tau_4(G, J)$
5	G	$\tau_4(G, J)$, $\tau_3(G, S)$, $\phi_L(L, G)$	G, J, L, S	$\tau_5(J, L, S)$
6	S	$\tau_5(J, L, S)$, $\phi_J(J, L, S)$	J, L, S	$\tau_6(J, L)$
7	L	$\tau_6(J, L)$	J, L	$\tau_7(J)$

Table 9.1 A run of variable elimination for the query $P(J)$

Step	Variable eliminated	Factors used	Variables involved	New factor
1	G	$\phi_G(G, I, D)$, $\phi_L(L, G)$, $\phi_H(H, G, J)$	G, I, D, L, J, H	$\tau_1(I, D, L, J, H)$
2	I	$\phi_I(I)$, $\phi_S(S, I)$, $\tau_1(I, D, L, S, J, H)$	S, I, D, L, J, H	$\tau_2(D, L, S, J, H)$
3	S	$\phi_J(J, L, S)$, $\tau_2(D, L, S, J, H)$	D, L, S, J, H	$\tau_3(D, L, J, H)$
4	L	$\tau_3(D, L, J, H)$	D, L, J, H	$\tau_4(D, J, H)$
5	H	$\tau_4(D, J, H)$	D, J, H	$\tau_5(D, J)$
6	C	$\tau_5(D, J)$, $\phi_C(C)$, $\phi_D(D, C)$	D, J, C	$\tau_6(D, J)$
7	D	$\tau_6(D, J)$	D, J	$\tau_7(J)$

Table 9.2 A different run of variable elimination for the query $P(J)$

9.3.1.3 Semantics of Factors

It is interesting to consider the semantics of the intermediate factors generated as part of this computation. In many of the examples we have given, they correspond to marginal or conditional probabilities in the network. However, although these factors often correspond to such probabilities, this is not always the case. Consider, for example, the network of figure 9.9a. The result of eliminating the variable X is a factor

$$\tau(A, B, C) = \sum_X P(X) \cdot P(A \mid X) \cdot P(C \mid B, X).$$

This factor does not correspond to any probability or conditional probability in this network. To understand why, consider the various options for the meaning of this factor. Clearly, it cannot be a conditional distribution where B is on the left hand side of the conditioning bar (for example, $P(A, B, C)$), as $P(B \mid A)$ has not yet been multiplied in. The most obvious candidate is $P(A, C \mid B)$. However, this conjecture is also false. The probability $P(A \mid B)$ relies heavily on the properties of the CPD $P(B \mid A)$; for example, if B is deterministically equal to A, $P(A \mid B)$ has a very different form than if B depends only very weakly on A. Since the CPD $P(B \mid A)$ was not taken into consideration when computing $\tau(A, B, C)$, it cannot represent the conditional probability $P(A, C \mid B)$. In general, we can verify that this factor

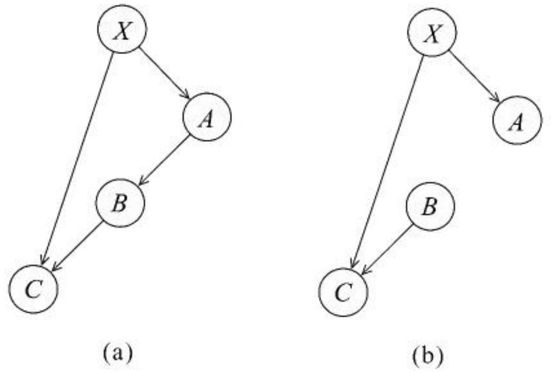


Figure 9.9 Understanding intermediate factors in variable elimination as conditional probabilities: (a) A Bayesian network where elimination does not lead to factors that have an interpretation as conditional probabilities. (b) A different Bayesian network where the resulting factor does correspond to a conditional probability.

does not correspond to any conditional probability expression in this network.

It is interesting to note, however, that the resulting factor does, in fact, correspond to a conditional probability $P(A, C \mid B)$, but *in a different network*: the one shown in figure 9.9b, where all CPDs except for B are the same. In fact, this phenomenon is a general one (see exercise 9.2).

9.3.2 Dealing with Evidence

It remains only to consider how we would introduce evidence. For example, assume we observe the value i^1 (the student is intelligent) and h^0 (the student is unhappy). Our goal is to compute $P(J \mid i^1, h^0)$. First, we reduce this problem to computing the unnormalized distribution $P(J, i^1, h^0)$. From this intermediate result, we can compute the conditional probability as in equation (9.1), by renormalizing by the probability of the evidence $P(i^1, h^0)$.

How do we compute $P(J, i^1, h^0)$? The key observation is proposition 4.7, which shows us how to view, as a Gibbs distribution, an unnormalized measure derived from introducing evidence into a Bayesian network. Thus, we can view this computation as summing out all of the entries in the *reduced factor*: $P[i^1 h^0]$ whose scope is $\{C, D, G, L, S, J\}$. This factor is no longer normalized, but it is still a valid factor.

factor reduction

Based on this observation, we can now apply precisely the same sum-product variable elimination algorithm to the task of computing $P(\boldsymbol{Y}, \boldsymbol{e})$. We simply apply the algorithm to the set of factors in the network, reduced by $\boldsymbol{E} = \boldsymbol{e}$, and eliminate the variables in $\mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$. The returned factor $\phi^*(\boldsymbol{Y})$ is precisely $P(\boldsymbol{Y}, \boldsymbol{e})$. To obtain $P(\boldsymbol{Y} \mid \boldsymbol{e})$ we simply renormalize $\phi^*(\boldsymbol{Y})$ by multiplying it by $\frac{1}{\alpha}$ to obtain a legal distribution, where α is the sum over the entries in our unnormalized distribution, which represents the probability of the evidence. To summarize, the algorithm for computing conditional probabilities in a Bayesian or Markov network is shown in algorithm 9.2.

We demonstrate this process on the example of computing $P(J, i^1, h^0)$. We use the same

Algorithm 9.2 Using Sum-Product-VE **for computing conditional probabilities**

```
    Procedure Cond-Prob-VE (
       $\mathcal{K}$,    // A network over $\mathcal{X}$
       $\boldsymbol{Y}$,    // Set of query variables
       $\boldsymbol{E} = \boldsymbol{e}$    // Evidence
    )
1      $\Phi \leftarrow$ Factors parameterizing $\mathcal{K}$
2      Replace each $\phi \in \Phi$ by $\phi[\boldsymbol{E} = \boldsymbol{e}]$
3      Select an elimination ordering $\prec$
4      $\boldsymbol{Z} \leftarrow\ = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$
5      $\phi^* \leftarrow$ Sum-Product-VE($\Phi, \prec, \boldsymbol{Z}$)
6      $\alpha \leftarrow \sum_{\boldsymbol{y} \in Val(\boldsymbol{Y})} \phi^*(\boldsymbol{y})$
7      return $\alpha, \phi^*$
```

Step	Variable eliminated	Factors used	Variables involved	New factor
1'	C	$\phi_C(C)$, $\phi_D(D, C)$	C, D	$\tau'_1(D)$
2'	D	$\phi_G[I = i^1](G, D)$, $\phi_I[I = i^1]()$, $\tau'_1(D)$	G, D	$\tau'_2(G)$
5'	G	$\tau'_2(G)$, $\phi_L(L, G)$, $\phi_H[H = h^0](G, J)$	G, L, J	$\tau'_5(L, J)$
6'	S	$\phi_S[I = i^1](S)$, $\phi_J(J, L, S)$	J, L, S	$\tau'_6(J, L)$
7'	L	$\tau'_6(J, L)$, $\tau'_5(J, L)$	J, L	$\tau'_7(J)$

Table 9.3 A run of sum-product variable elimination for $P(J, i^1, h^0)$

elimination ordering that we used in table 9.1. The results are shown in table 9.3; the step numbers correspond to the steps in table 9.1. It is interesting to note the differences between the two runs of the algorithm. First, we notice that steps (3) and (4) disappear in the computation with evidence, since I and H do not need to be eliminated. More interestingly, by not eliminating I, we avoid the step that correlates G and S. In this execution, G and S never appear together in the same factor; they are both eliminated, and only their end results are combined. Intuitively, G and S are conditionally independent given I; hence, observing I renders them independent, so that we do not have to consider their joint distribution explicitly. Finally, we notice that $\phi_I[I = i^1] = P(i^1)$ is a factor over an empty scope, which is simply a number. It can be multiplied into any factor at any point in the computation. We chose arbitrarily to incorporate it into step (2′). Note that if our goal is to compute a conditional probability given the evidence, and not the probability of the evidence itself, we can avoid multiplying in this factor entirely, since its effect will disappear in the renormalization step at the end.

network polynomial

Box 9.A — Concept: The Network Polynomial. *The* network polynomial *provides an interesting and useful alternative view of variable elimination. We begin with describing the concept for the case of a Gibbs distribution parameterized via a set of full table factors* Φ. *The polynomial* f_Φ

is defined over the following set of variables:

- *For each factor $\phi_c \in \Phi$ with scope $\boldsymbol{X}_c$, we have a variable $\theta_{\boldsymbol{x}_c}$ for every $\boldsymbol{x}_c \in Val(\boldsymbol{X}_c)$.*
- *For each variable X_i and every value $x_i \in Val(X_i)$, we have a binary-valued variable λ_{x_i}.*

In other words, the polynomial has one argument for each of the network parameters and for each possible assignment to a network variable. The polynomial f_Φ is now defined as follows:

$$f_\Phi(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \sum_{x_1,\ldots,x_n} \left(\prod_{\phi_c \in \Phi} \theta_{\boldsymbol{x}_c} \cdot \prod_{i=1}^{n} \lambda_{x_i} \right). \tag{9.7}$$

Evaluating the network polynomial is equivalent to the inference task. In particular, let $\boldsymbol{Y} = \boldsymbol{y}$ be an assignment to some subset of network variables; define an assignment $\boldsymbol{\lambda}^{\boldsymbol{y}}$ as follows:

- *for each $Y_i \in \boldsymbol{Y}$, define $\lambda^{\boldsymbol{y}}_{y_i} = 1$ and $\lambda^{\boldsymbol{y}}_{y'_i} = 0$ for all $y'_i \neq y_i$;*
- *for each $Y_i \notin \boldsymbol{Y}$, define $\lambda^{\boldsymbol{y}}_{y_i} = 1$ for all $y_i \in Val(Y_i)$.*

With this definition, we can now show (exercise 9.4a) that:

$$f_\Phi(\boldsymbol{\theta}, \boldsymbol{\lambda}^{\boldsymbol{y}}) = \tilde{P}_\Phi(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{\theta}). \tag{9.8}$$

The derivatives of the network polynomial are also of significant interest. We can show (exercise 9.4b) that

$$\frac{\partial f_\Phi(\boldsymbol{\theta}, \boldsymbol{\lambda}^{\boldsymbol{y}})}{\partial \lambda_{x_i}} = \tilde{P}_\Phi(x_i, \boldsymbol{y}_{-i} \mid \boldsymbol{\theta}), \tag{9.9}$$

where $\boldsymbol{y}_{-i}$ is the assignment in $\boldsymbol{y}$ to all variables other than X_i. We can also show that

$$\frac{\partial f_\Phi(\boldsymbol{\theta}, \boldsymbol{\lambda}^{\boldsymbol{y}})}{\partial \theta_{\boldsymbol{x}_c}} = \frac{\tilde{P}_\Phi(\boldsymbol{y}, \boldsymbol{x}_c \mid \boldsymbol{\theta})}{\theta_{\boldsymbol{x}_c}}; \tag{9.10}$$

sensitivity analysis

this fact is proved in lemma 19.1. These derivatives can be used for various purposes, including retracting or modifying evidence in the network (exercise 9.4c), and sensitivity analysis *— computing the effect of changes in a network parameter on the answer to a particular probabilistic query (exercise 9.5).*

Of course, as defined, the representation of the network polynomial is exponentially large in the number of variables in the network. However, we can use the algebraic operations performed in a run of variable elimination to define a network polynomial that has precisely the same complexity as the VE run. More interesting, we can also use the same structure to compute efficiently all of the derivatives of the network polynomial, relative both to the λ_i and the $\theta_{\boldsymbol{x}_c}$ (see exercise 9.6).