An Introduction to Probabilistic Graphical Models

Michael I. Jordan
University of California, Berkeley

June 30, 2003

Chapter 3

The Elimination Algorithm

In this chapter we discuss the problem of computing conditional and marginal probabilities in graphical models—the problem of *probabilistic inference*. Building on the ideas in Chapter 2, we show how the conditional independencies encoded in a graph can be exploited for efficient computation of conditional and marginal probabilities.

We take a very concrete approach in the current chapter, basing the presentation on a simple "elimination algorithm" for probabilistic inference. This algorithm applies equally well to directed and undirected graphs. It requires little formal machinery to describe and to analyze. On the other hand, the algorithm has its limitations, and is not our final word on the inference problem. But it is a good place to start.

3.1 Probabilistic inference

Let E and F be disjoint subsets of the node indices of a graphical model, such that X_E and X_F are disjoint subsets of the random variables in the domain. Our goal is to calculate $p(x_F \,|\, x_E)$ for arbitrary subsets E and F. This is the general *probabilistic inference problem* for graphical models (directed or undirected).

We begin by focusing on directed graphs. Almost all of our work, however, will transfer to undirected graphs with little or no change. Our subsequent treatment of undirected models in Section 3.1.3 will be short and sweet.

Throughout the chapter we limit ourselves to the probability of calculating the conditional probability of a single node X_F—which we refer to as the "query node"—given an arbitrary set of nodes X_E. This is a limitation of the simple elimination algorithm that we discuss in this chapter, and is not a limitation of the more general algorithms that we discuss in later chapters.

Graphically we indicate the set of conditioning variables by shading the corresponding nodes in the graph. Thus, the dark shading in Figure 3.1 indicates the nodes (indexed by E) on which we condition. We will often refer to these nodes as the *evidence nodes*. The unshaded nodes (indexed by F) are the nodes for which we wish to compute conditional probabilities. Finally, the lightly shaded nodes, indexed by $R = V \backslash (E \cup F)$, are the nodes that must be marginalized out of the joint probability so that we can focus on the conditional, $p(x_F \,|\, x_E)$, of interest. Thus, symbolically, we

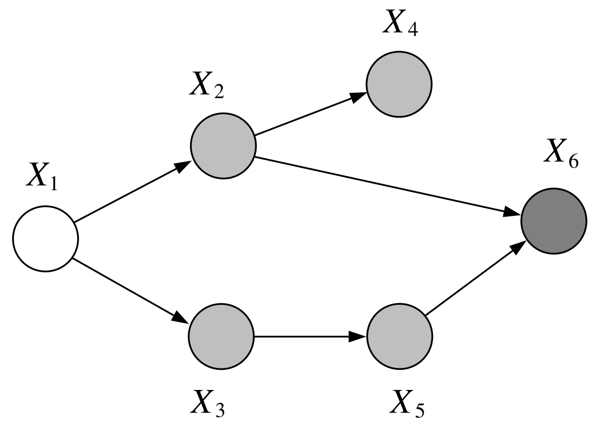

Figure 3.1: The dark shaded node, X_6, is the node on which we condition, the lightly shaded nodes, $\{X_2, X_3, X_4, X_5\}$, are nodes that are marginalized over, and the unshaded node, X_1 is the node for which we wish to calculate conditional probabilities. Thus, for this example, we have $E = \{6\}$, $F = \{1\}$, and $R = \{2, 3, 4, 5\}$.

must compute the marginal:

$$p(x_E, x_F) = \sum_{x_R} p(x_E, x_F, x_R), \tag{3.1}$$

which can be further marginalized to yield $p(E)$:

$$p(x_E) = \sum_{x_F} p(x_E, x_F), \tag{3.2}$$

from which we obtain the conditional probability:

$$p(x_F \,|\, x_E) = \frac{p(x_E, x_F)}{p(x_E)}. \tag{3.3}$$

We will be interested in finding effective computational methods for making these calculations.

A special case of the general problem is worth noting. Consider the case of just two nodes, X and Y, as shown in Figure 3.2(a). This model is specified in terms of the distributions $p(x)$ and $p(y \,|\, x)$, reflecting the arrow from X to Y. Suppose that we condition on X, as shown in Figure 3.2(b), and wish to calculate the probability of Y. This "calculation" is simply a table lookup using $p(y \,|\, x)$. On the other hand, suppose that we condition on Y and wish to calculate the probability of X, as indicated in Figure 3.2(c). This is achieved via an application of Bayes rule:

$$p(x \,|\, y) = \frac{p(y \,|\, x)p(x)}{p(y)}. \tag{3.4}$$

where the denominator is calculated as follows:

$$p(y) = \sum_{x} p(y \,|\, x)p(x). \tag{3.5}$$

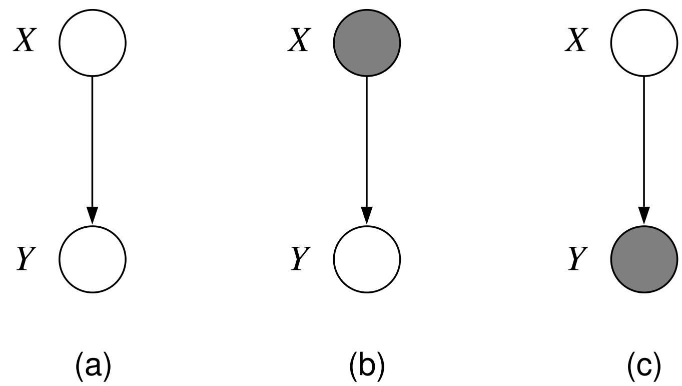

Figure 3.2: (a) A two-node model. (b) Conditioning on X involves a simple evaluation of $p(y \mid x)$. (c) Conditioning on Y requires the use of Bayes rule.

Can we find an efficient extension of these familiar ideas to general graphs?

The summations in Eq. (3.1) and Eq. (3.2) should give us pause. The summation $\sum_{x_R}$ expands into a sequence of summations, one for each of the random variables indexed by R. If each such random variable can take on k values, and there are $|R|$ variables, we obtain $k^{|R|}$ terms in our summation. A similar statement applies to the summation $\sum_{x_F}$. With $|F|$ and $|R|$ in the dozens or hundreds in typical cases, naive summation is infeasible.

We need to take advantage of the factorization offered by the definition of the joint probability. If we do not take advantage of the factorization we will be in trouble performing even a single summation, much less a sequence of summations. Consider summing $p(x_1, x_2, \ldots, x_6)$ with respect to x_6, where we naively represent the joint probability as a table of size k^6. (Recall that k is the number of values that each variable x_i can take on, assumed independent of i for simplicity.) Given that we must perform the sum for each value of the variables $\{x_1, x_2, \ldots, x_5\}$, we see that we must perform $O(k^6)$ operations to do a single sum (essentially, we must touch each entry in the table). To reduce the computational complexity let us instead represent the joint probability in its factored form (cf. Eq. (2.3)) and exploit the distributive law:

$$
\begin{aligned}
p(x_1, x_2, \ldots, x_5) &= \sum_{x_6} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3)p(x_6 \mid x_2, x_5) && (3.6)\\
&= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3) \sum_{x_6} p(x_6 \mid x_2, x_5). && (3.7)
\end{aligned}
$$

The summation over x_6 is now applied to $p(x_6 \mid x_2, x_5)$, a table of size k^3. We have reduced the operation count from $O(k^6)$ to $O(k^3)$, a significant improvement.[1]

Successive summations also take advantage of the factorization. A summation over, say, x_5, can also be moved along the chain of factors until it encounters a factor involving x_5. If each such summation is of reduced complexity, say $O(k^r)$ for some r, then the result is an algorithm that

[1] Of course this sum is unity by the definition of conditional probability, and thus we don't actually have to perform any operations at all, but let us pretend not to know that.

scales as $O(nk^r)$ instead of $O(k^n)$. Of course, the summations create intermediate factors that may link variables, making it not entirely clear whether or not we can keep r small. It is here that graphical methods are helpful. We can determine the parameter r by a graph-theoretic algorithm.

Let us introduce the basic ideas in the context of an example. Referring to the graph in Figure 3.1, let us condition on the event $\{X_6 = x_6\}$ and calculate the conditional probability $p(x_1 \,|\, x_6)$.

A point to note at the outset is that x_6 is a fixed constant in this calculation and does not contribute to the computational complexity of the calculation. Thus, while the table representing $p(x_6 \,|\, x_2, x_5)$ is nominally a three-dimensional table, the observation of X_6 involves taking a two-dimensional slice of this table. Unfortunately our notation is ambiguous in this regard; we have been using "x_6" as a variable that ranges over the possible values of X_6. In particular it is meaningful to sum over "x_6." In the remainder of this section, to avoid confusion, we refer to a particular fixed value of X_6 as "$\bar{x}_6$." Thus, we wish to compute $p(x_1 \,|\, \bar{x}_6)$, for any x_1 and for a particular $\bar{x}_6$.

We begin by computing the probability $p(x_1, \bar{x}_6)$ by summing over $\{x_2, x_3, x_4, x_5\}$. We introduce some notation to refer to intermediate factors that arise when performing these sums. In particular, let $m_i(x_{S_i})$ denote the expression that arises from performing the sum $\sum_{x_i}$, where x_{S_i} are the variables, other than x_i, that appear in the summand. Thus we have:

$$\begin{aligned} p(x_1, \bar{x}_6) &= \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} p(x_1) p(x_2 \,|\, x_1) p(x_3 \,|\, x_1) p(x_4 \,|\, x_2) p(x_5 \,|\, x_3) p(\bar{x}_6 \,|\, x_2, x_5) && (3.8) \\ &= p(x_1) \sum_{x_2} p(x_2 \,|\, x_1) \sum_{x_3} p(x_3 \,|\, x_1) \sum_{x_4} p(x_4 \,|\, x_2) \sum_{x_5} p(x_5 \,|\, x_3) p(\bar{x}_6 \,|\, x_2, x_5) && (3.9) \\ &= p(x_1) \sum_{x_2} p(x_2 \,|\, x_1) \sum_{x_3} p(x_3 \,|\, x_1) \sum_{x_4} p(x_4 \,|\, x_2) m_5(x_2, x_3) && (3.10) \end{aligned}$$

where we define $m_5(x_2, x_3) \triangleq \sum_{x_5} p(x_5 \,|\, x_3) p(\bar{x}_6 \,|\, x_2, x_5)$. (Note that to simplify notation we do not indicate explicitly the dependence of this term on the constant $\bar{x}_6$). Computing $m_5(x_2, x_3)$ has eliminated X_5 from further consideration in the computation. As we will see later, this algebraic notion of "elimination" corresponds to a graphical notion of elimination in which the node X_5 is removed from the graph. We continue the derivation:

$$\begin{aligned} p(x_1, \bar{x}_6) &= p(x_1) \sum_{x_2} p(x_2 \,|\, x_1) \sum_{x_3} p(x_3 \,|\, x_1) m_5(x_2, x_3) \sum_{x_4} p(x_4 \,|\, x_2) && (3.11) \\ &= p(x_1) \sum_{x_2} p(x_2 \,|\, x_1) m_4(x_2) \sum_{x_3} p(x_3 \,|\, x_1) m_5(x_2, x_3). && (3.12) \end{aligned}$$

Of course, $m_4(x_2) \triangleq \sum_{x_4} p(x_4 \,|\, x_2)$ is equal to one by definition, and in practice we would not do this sum, but let us be systematic and keep the term in our calculations. Finally, we have:

$$\begin{aligned} p(x_1, \bar{x}_6) &= p(x_1) \sum_{x_2} p(x_2 \,|\, x_1) m_4(x_2) m_3(x_1, x_2) && (3.13) \\ &= p(x_1) m_2(x_1). && (3.14) \end{aligned}$$

From this result we can also obtain the probability $p(\bar{x}_6)$ by taking an additional sum over x_1:

$$p(\bar{x}_6) = \sum_{x_1} p(x_1) m_2(x_1), \tag{3.15}$$

and the desired conditional is obtained by dividing Eq. (3.14) into Eq. (3.15):

$$p(x_1 \,|\, \bar{x}_6) = \frac{p(x_1) m_2(x_1)}{\sum_{x_1} p(x_1) m_2(x_1)}. \tag{3.16}$$

Alternatively we can view $p(x_1, \bar{x}_6)$ in Eq. (3.14) as an unnormalized representation of the conditional probability $p(x_1 \,|\, \bar{x}_6)$—recall once again that $\bar{x}_6$ is a fixed constant. Thus we obtain the conditional by normalization, where the normalization constant is given by Eq. (3.15).

Lying behind this flurry of algebra is a simple general algorithm for computing marginal probabilities. We present this algorithm in Section 3.1.2. First, however, we set the stage for the general algorithm with some preparatory remarks on conditioning.

3.1.1 Conditioning

To provide a simple exposition of the general elimination algorithm in Section 3.1.2, and also to simplify our exposition of inference algorithms presented in later chapters, it is useful to make use of a notational trick in which conditioning is viewed as a summation. This trick will allow us to treat marginalization and conditioning as formally equivalent, and will make it easier to bring the key operations of the inference algorithms into focus.

Let X_i be an evidence node whose observed value is $\bar{x}_i$. To capture the fact that X_i is fixed at the value $\bar{x}_i$, we define an *evidence potential*, $\delta(x_i, \bar{x}_i)$, a function whose value is one if $x_i = \bar{x}_i$ and zero otherwise. The evidence potential allows us to turn evaluations into sums: To evaluate a function $g(x_i)$ at $\bar{x}_i$ we multiply $g(x_i)$ by $\delta(x_i, \bar{x}_i)$ and sum over x_i:

$$g(\bar{x}_i) = \sum_{x_i} g(x_i) \delta(x_i, \bar{x}_i), \tag{3.17}$$

a trick that also extends to multivariate functions with x_i as one of the arguments. In particular, returning to the example from the previous section, we can express the evaluation of $p(x_6 \,|\, x_2, x_5)$ at $\bar{x}_6$ as follows:

$$m_6(x_2, x_5) = \sum_{x_6} p(x_6 \,|\, x_2, x_5) \delta(x_6, \bar{x}_6), \tag{3.18}$$

where $m_6(x_2, x_5)$ is nothing but $p(\bar{x}_6 \,|\, x_2, x_5)$.

In general, let E be the set of nodes whose values are to be conditioned on. That is, for a specific configuration $\bar{x}_E$, we wish to compute $p(x_F \,|\, \bar{x}_E)$. Formally, we achieve this as follows. Define the *total evidence potential*:

$$\delta(x_E, \bar{x}_E) \triangleq \prod_{i \in E} \delta(x_i, \bar{x}_i), \tag{3.19}$$

a function that is equal to one if $x_E = \bar{x}_E$ and is equal to zero otherwise. Using this potential, we can obtain both the numerator and the denominator of the conditional probability $p(x_F \,|\, \bar{x}_E)$ by summation. Thus:

$$p(x_F, \bar{x}_E) = \sum_{x_E} p(x_F, x_E)\delta(x_E, \bar{x}_E) \tag{3.20}$$

and:

$$p(\bar{x}_E) = \sum_{x_F} \sum_{x_E} p(x_F, x_E)\delta(x_E, \bar{x}_E). \tag{3.21}$$

This suggests that it may be useful to define:

$$p^E(x) \triangleq p(x)\delta(x_E, \bar{x}_E) \tag{3.22}$$

as a generalized measure that represents conditional probability with respect to E. By formally "marginalizing" this measure with respect to x_E, we evaluate $p(x)$ at $X_E = \bar{x}_E$, and obtain $p(x_F, \bar{x}_E)$, an unnormalized version of the conditional probability $p(x_F \,|\, \bar{x}_E)$. Moreover, by marginalizing over x, we obtain the total "mass" $p(\bar{x}_E)$.

This tactic is particularly natural in the case of undirected graphs, where multiplication by an evidence potential $\delta(x_i, \bar{x}_i)$ can be implemented by simply redefining the local potentials $\psi(x_i)$ for $i \in E$. Thus, we define:

$$\psi_i^E(x_i) \triangleq \psi_i(x_i)\delta(x_i, \bar{x}_i), \tag{3.23}$$

for $i \in E$. Leaving all other clique potentials unchanged, that is, letting $\psi_C^E(x_C) \triangleq \psi_C(x_C)$, for $C \notin \{\{i\} : i \in E\}$, we obtain the desired unnormalized representation:

$$p^E(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{X_C}^E(x_C). \tag{3.24}$$

Moreover, since we are working with an unnormalized representation, we may as well drop the $1/Z$ factor, and simply work with $\prod_{C \in \mathcal{C}} \psi_{X_C}^E(x_C)$ as an unnormalized representation of conditional probability.

It should be clear that the use of evidence potentials is merely a piece of formal trickery that will (turn out to) simplify our description of various inference algorithms. In practice we would not actually perform the sum over a function that we know to be zero over most of the sample space, but rather we would take "slices" of the appropriate probabilities or potentials. Thus, in evaluating $p(x_6 \,|\, x_2, x_5)$ at $X_6 = \bar{x}_6$, while formally we can view ourselves as multiplying by $\delta(x_6, \bar{x}_6)$ and summing over x_6, algorithmically we would simply take the appropriate two-dimensional slice of the three-dimensional table representing $p(x_6 \,|\, x_2, x_5)$.

3.1.2 Elimination and directed graphs

In this section we describe a general algorithm for performing probabilistic inference in directed graphical models.

At each step of the algorithm, we perform a sum over a product of functions. The functions that can appear in such sums include the original local conditional probabilities, $p(x_i \,|\, x_{\pi_i})$, the evidence potentials, $\delta(x_i, \bar{x}_i)$, and the intermediate factors, $m_i(x_{S_i})$, generated by previous sums. All of these functions are defined on local subsets of nodes, and we use the generic term "potential" to refer to all of them.[2] Thus our algorithm will involve taking sums over products of potential functions.

The algorithm works as follows (see Figure 3.3 for a summary). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an evidence set E, and a query node F, we first choose an elimination ordering I such that F appears last in the ordering.[3] Throughout the algorithm we maintain an *active list* of potential functions. The active list is initialized to hold the local conditional probabilities, $p(x_i \,|\, x_{\pi_i})$, for $i \in \mathcal{V}$, and the evidence potentials, $\delta(x_i, \bar{x}_i)$, for $i \in E$. At each step of the algorithm, we find all those potentials on the active list that reference the next node (call it X_i) in the elimination ordering I. These potential functions are removed from the active list. We take the product of these functions and sum this product with respect to x_i. This defines a new intermediate factor, $m_i(x_{S_i})$, where x_{S_i} are the variables (other than x_i) that appear in the summand. This intermediate factor is added to the active list. We then proceed to the next node in the elimination ordering.

Note that we have introduced the notation $T_i = \{i\} \cup S_i$ in the description of the algorithm in Figure 3.3. The subset T_i indexes the set of all variables that appear in the summand of the operator $\sum_{x_i}$. We give a graph-theoretic interpretation of T_i later in the chapter.

The algorithm terminates when we arrive at the final node in the elimination ordering, the query node X_F. The product of potentials on the active list at this point defines the (unnormalized) conditional probability, $p(x_F, \bar{x}_E)$. Summing this product over x_F yields the normalization factor $p(\bar{x}_E)$.

Let us now return to the example in Section 3.1 and show how the steps of ELIMINATE correspond to the steps in the algebraic calculation in that section. The evidence node in this example is X_6 and the query node is X_1. We choose the elimination ordering $I = (6, 5, 4, 3, 2, 1)$, in which the query node appears last.

We begin by placing the local conditional probabilities, $\{p(x_1), \ldots, p(x_6 \,|\, x_2, x_5)\}$, on the active list. We also place $\delta(x_6, \bar{x}_6)$ on the active list.

We first eliminate node X_6. The potential functions on the active list that reference x_6 are $p(x_6 \,|\, x_2, x_5)$ and $\delta(x_6, \bar{x}_6)$. Thus we have $\phi_6(x_2, x_5, x_6) = p(x_6 \,|\, x_2, x_5)\delta(x_6, \bar{x}_6)$. Summing this expression with respect to x_6 yields $m_6(x_2, x_5) = p(\bar{x}_6 \,|\, x_2, x_5)$. We place this potential on the active list, having removed $p(x_6 \,|\, x_2, x_5)$ and $\delta(x_6, \bar{x}_6)$. We have simply evaluated $p(x_6 \,|\, x_2, x_5)$ at $\bar{x}_6$.

We now eliminate X_5. The potentials on on the active list that reference x_5 are $p(x_5 \,|\, x_3)$

[2] The reader may be concerned that we are using the term "potential" somewhat loosely here. In particular we are using it in the context of directed graphs and in the context of subsets that may not be cliques; this usage clashes with the definition of "potential" in Chapter ??. We hope that the reader will forgive the seeming abuse of terminology. It is worth noting, however, that the "potentials" discussed in this section are in fact honest-to-goodness potentials, but not with respect to G. Rather they are potentials on the cliques of a different graph, a graph known as the *moral graph* G^m. This point will be clarified in Section ?? below.

[3] We will not discuss the choice of elimination ordering in this chapter, but instead will defer this (non-trivial) problem until Chapter 17, where it will arise in a more general way in the context of the junction tree algorithm. For now, let the ordering I be arbitrary, under the constraint that F appears last. We might encourage the reader, however, to start to ponder how to characterize good elimination orderings. Some useful food for thought in this regard will be provided in Section ?? below.

```
ELIMINATE(𝒢, E, F)
    INITIALIZE(𝒢, F)
    EVIDENCE(E)
    UPDATE(𝒢)
    NORMALIZE(F)

INITIALIZE(𝒢, F)
    choose an ordering I such that F appears last
    for each node X_i in 𝒱
        place p(x_i | x_{π_i}) on the active list
    end

EVIDENCE(E)
    for each i in E
        place δ(x_i, x̄_i) on the active list
    end

UPDATE(𝒢)
    for each i in I
        find all potentials from the active list that reference x_i and remove them from the active list
        let φ_i(x_{T_i}) denote the product of these potentials
        let m_i(x_{S_i}) = Σ_{x_i} φ_i(x_{T_i})
        place m_i(x_{S_i}) on the active list
    end

NORMALIZE(F)
    p(x_F | x̄_E) ← φ_F(x_F) / Σ_{x_F} φ_F(x_F)
```

Figure 3.3: The ELIMINATE algorithm for probabilistic inference on directed graphs.

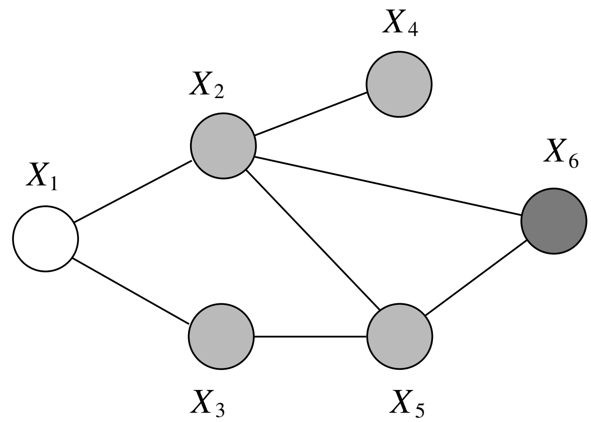

Figure 3.4: The dark shaded node, X_6, is the nodes on which we condition, the lightly shaded nodes, $\{X_2, X_3, X_4, X_5\}$, are the nodes that are marginalized over, and the unshaded node, X_1, is the node for which we wish to calculate conditional probabilities.

and $m_6(x_2, x_5)$. We remove them, and define the product $\phi_5(x_2, x_3, x_5)$. Summing over x_5 yields $m_5(x_2, x_3)$ (cf. Eq. (3.11)).

The only potential that references X_4 is $p(x_4 \mid x_2)$. The elimination of X_4 thus involves summing $p(x_4 \mid x_2)$ with respect to x_4 to obtain the factor $m_4(x_2)$. This factor is identically one and in practice we would not bother computing it.

Eliminating X_3 involves taking the sum over $\phi_3(x_1, x_2, x_3) = p(x_3 \mid x_1)m_5(x_2, x_3)$ to yield $m_3(x_1, x_2)$ and we are now at Eq. (3.13) in the earlier derivation.

We now eliminate X_2 to obtain $\phi_1(x_1) = p(x_1)m_2(x_1)$, which is the "unnormalized conditional probability," $p(x_1, \bar{x}_6)$. Eliminating X_1 yields $m_1 = \sum_{x_1} \phi_1(x_1)$, which is the normalization factor, $p(\bar{x}_6)$.

3.1.3 Elimination and undirected graphs

In the case of directed models, we have shown that the problem of calculating conditional probabilities can be usefully viewed in terms of a simple elimination algorithm. The same perspective applies to undirected models, and indeed the entire Eliminate algorithm from Figure 3.3 goes through without essential change to the undirected case.

The only change needed to handle the undirected case occurs in the Initialize procedure, where instead of using local conditional probabilities we initialize the active list to contain the potentials $\{\psi_{X_C}(x_C)\}$.

Let us briefly consider an example. Paralleling the example from Section 3.1.2 we calculate the probability $p(x_1 \mid \bar{x}_6)$ for the graph in Figure 3.4. We represent the joint probability on the graph via potential functions on the cliques $\{X_1, X_2\}$ $\{X_1, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_5\}$, and $\{X_2, X_5, X_6\}$.

We first calculate the probability $p(x_1, \bar{x}_6)$. To simplify the presentation we drop the subscript in the $\psi_{X_C}(x_C)$ notation, relying on the argument to the function to make it clear which potential function is being referred to. Also we again make use of the notation $m_i(x_{S_i})$ to denote the intermediate factor that results from the summation over x_i. We have:

$$
\begin{aligned}
p(x_1, \bar{x}_6) &= \frac{1}{Z} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \sum_{x_6} \psi(x_1,x_2)\psi(x_1,x_3)\psi(x_2,x_4)\psi(x_3,x_5)\psi(x_2,x_5,x_6)\delta(x_6,\bar{x}_6) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1,x_2) \sum_{x_3} \psi(x_1,x_3) \sum_{x_4} \psi(x_2,x_4) \sum_{x_5} \psi(x_3,x_5) \sum_{x_6} \psi(x_2,x_5,x_6)\delta(x_6,\bar{x}_6) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1,x_2) \sum_{x_3} \psi(x_1,x_3) \sum_{x_4} \psi(x_2,x_4) \sum_{x_5} \psi(x_3,x_5) m_6(x_2,x_5) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1,x_2) \sum_{x_3} \psi(x_1,x_3) m_5(x_2,x_3) \sum_{x_4} \psi(x_2,x_4) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1,x_2) m_4(x_2) \sum_{x_3} \psi(x_1,x_3) m_5(x_2,x_3) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1,x_2) m_4(x_2) m_3(x_1,x_2) \\
&= \frac{1}{Z} m_2(x_1). \qquad (3.25)
\end{aligned}
$$

Marginalizing further over x_1 yields:

$$p(\bar{x}_6) = \frac{1}{Z} \sum_{x_1} m_2(x_1), \qquad (3.26)$$

and we calculate the desired conditional as:

$$p(x_1 \mid \bar{x}_6) = \frac{m_2(x_1)}{\sum_{x_1} m_2(x_1)}, \qquad (3.27)$$

where the normalization factor Z cancels.

Note that the calculation in the example is formally identical to the corresponding calculation for directed graphs. Note, however, that the sum $m_4(x_2)$, which earlier could be omitted, no longer necessarily sums to one and must be explicitly carried along in the calculation.

Finally, a remark on the computation of marginal probabilities $p(x_i)$. For a marginal probability the normalization factor Z does not cancel, and must be calculated explicitly. Just as in the other calculations in this section, however, the calculation of Z is a summation over the unnormalized representation of the joint probability, and indeed it is simply a summation over *all* of the variables. To obtain the marginal $p(x_i)$, we would define an elimination ordering in which x_i is the final variable, and then normalize the result to calculate Z and obtain the marginal.

In the directed case, a variable that is parentless has its marginal represented explicitly in the parameterization of the graphical model and no calculation is needed. In general, nodes that are downstream from a target node can simply be deleted, and marginalization involves an inference calculation involving the ancestors of the node. The worst case is a leaf node. In the undirected case, there is no notion of "ancestor," and essentially all nodes are worst case. On the other hand, once Z is calculated from a particular elimination ordering, it can be used to normalize other marginal probabilities.

UndirectedGraphEliminate($\mathcal{G}, I$)
 for each node X_i in I
  connect all of the remaining neighbors of X_i
  remove X_i from the graph
 end

Figure 3.5: A simple greedy algorithm for eliminating nodes in an undirected graph $\mathcal{G}$.

3.2 Graph elimination

The simple Eliminate algorithm captures the key algorithmic operation underlying probabilistic inference—that of taking a sum over a product of potential functions. What can we say about the overall computational complexity of the algorithm? In particular, how can we control the "size" of the summands that appear in the sequence of summation operations?

In this section, we show that questions regarding the computational complexity of the Eliminate algorithm can be reduced to purely graph-theoretic considerations. This graphical interpretation will also provide hints about how to design improved inference algorithms that overcome the limitations of Eliminate.

3.2.1 A graph elimination algorithm

Let us put aside marginalization and probabilistic inference for a moment, and concentrate solely on graph-theoretic manipulations. We describe a simple procedure that eliminates nodes in a graph. As will become clear, this procedure captures the graph-theoretic operations underlying Eliminate.

We begin by describing a node-elimination algorithm for undirected graphs, making the link to directed graphs shortly.

Assume that we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an elimination ordering I. We describe a simple graph-theoretic algorithm that successively eliminates the nodes of $\mathcal{G}$. In particular, at each step, the algorithm eliminates the next node in the ordering I, where "eliminate" means removing the node from the graph and connecting the (remaining) neighbors of the node. The algorithm, which we refer to as UndirectedGraphEliminate, is presented in Figure 3.5.

We will be interested in the *reconstituted graph*; the graph $\tilde{\mathcal{G}} = (\mathcal{V}, \tilde{\mathcal{E}})$, whose edge set $\tilde{\mathcal{E}}$ is a superset of $\mathcal{E}$, incorporating all of the original edges $\mathcal{E}$, as well as any new edges created during a run of UndirectedGraphEliminate.

Consider in particular the graph in Figure 3.6(a) and the elimination ordering $(6, 5, 4, 3, 2, 1)$. Let us run through the graphical elimination procedure. Starting with node X_6 we first connect its neighbors, adding an edge between X_2 and X_5, as shown in Figure 3.6(b). We then remove X_6, which yields Figure 3.6(c). Moving to X_5, we connect its neighbors, X_2 and X_3, and remove X_5, which yields Figure 3.6(d). The procedure continues in Figure 3.6(e)–(g), culminating in a graph with the single node X_1, which is then removed yielding the empty graph.

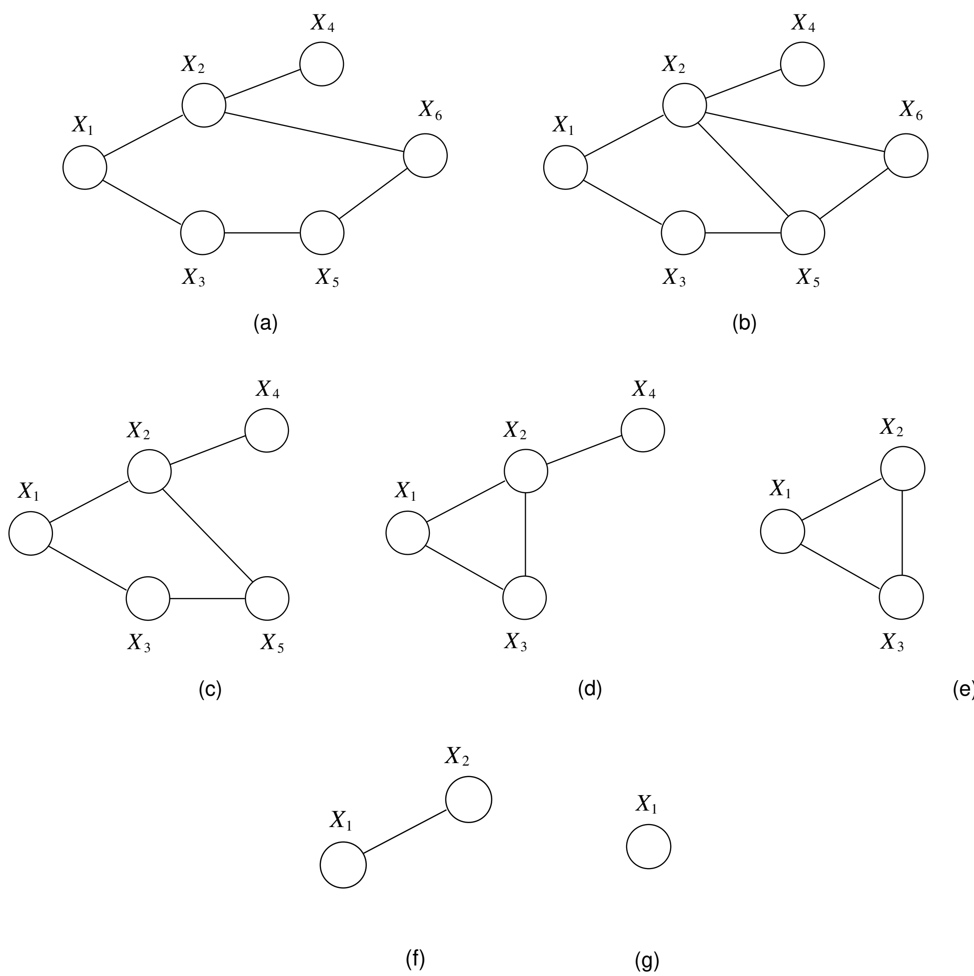

Figure 3.6: A run of the elimination algorithm under the elimination ordering $(6, 5, 4, 3, 2, 1)$. The original graph is shown in (a).

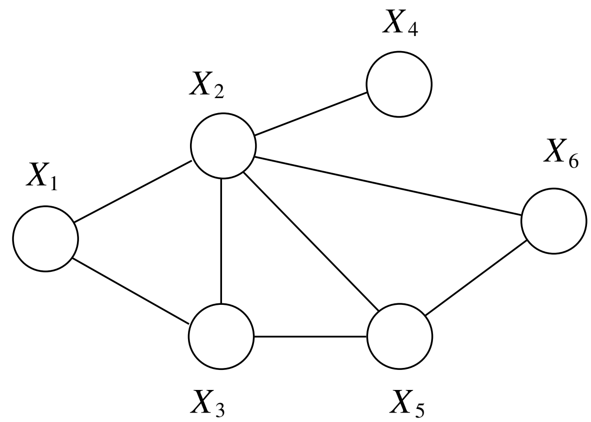

Figure 3.7: The reconstituted graph, showing the edges that were added during the elimination process.

Figure 3.7 shows the reconstituted graph, where we have retained the edges that were created during the elimination procedure (in particular, the edges between X_2 and X_3 and between X_2 and X_5). This graph turns out to have some important graph-theoretic properties, properties which underly the comprehensive theory of inference that will be the subject of Chapter 17.[4] For current purposes, however, the relevant properties of the graph can be captured by recording the *elimination cliques* of the graph. In particular, each time we remove a node X_i in the second step of the algorithm, let us record the collection of nodes that are the neighbors of X_i at that moment, including X_i itself. These nodes form a fully-connected subset of nodes by virtue of the first step of the algorithm; that is, they form a clique. We denote the set of indices of the nodes in this clique as T_i. Thus, in our example, $T_6 = \{2, 5, 6\}$ and $T_5 = \{2, 3, 5\}$. (Note that index 6 does not appear in T_5 because X_6 has already been eliminated when we process node X_5).

3.2.2 Graph elimination and marginalization

When we perform a marginalization operation, removing a random variable from a joint distribution, we perform a sum over the product of all factors that depend on that random variable. This couples together all of the other random variables that appear in those factors. Thus, for example, summing the product $\psi(x_3, x_5)m_6(x_2, x_5)$ with respect to x_5 creates an intermediate factor that involves x_2 and x_3. This new factor does not in general factorize with respect to x_2 and x_3; thus, we have an *induced dependency* between x_2 and x_3. Subsequent operations will have to treat x_2 and x_3 together. UNDIRECTEDGRAPHELIMINATE makes this coupling explicit, by linking the neighbors of the node being summed over.

In general, as we now show, the elimination cliques in UNDIRECTEDGRAPHELIMINATE are the graph-theoretic counterparts of the sets of variables on which summations operate in probabilistic

[4]For readers who cannot bear the wait, the key property of the reconstituted graph is that it is a *triangulated graph*; indeed, our elimination procedure is a simple algorithm for triangulating a graph.

inference using ELIMINATE.

Consider the argument x_{T_i} to the function $\phi_i(x_{T_i})$ in ELIMINATE, the function which is the summand for the operator $\sum_{x_i}$. As our notation suggests, the variables referenced by ϕ_i are exactly those in the elimination clique created by UNDIRECTEDGRAPHELIMINATE upon elimination of X_i. To see this, note that any potential removed from the active list by ELIMINATE (when summing over x_i) must reference x_i. Now consider any other variable x_j referenced by $\phi_i(x_{T_i})$. We want to show that X_i and X_j must be neighbors in the graph constructed by UNDIRECTEDGRAPHELIMINATE. There are two cases to consider, corresponding to the two kinds of potentials that can link variables: (1) If the potential is one of the original potentials $\psi_C(x_C)$, then X_j is necessarily linked to X_i, because C is a clique (by definition). (2) If x_i and x_j appear together in an intermediate factor $m_k(x_{S_k})$, then this term was created by the elimination of an earlier node X_k. At the moment of eliminating X_k, UNDIRECTEDGRAPHELIMINATE must have linked the nodes X_i and X_j. Thus, in either case, X_j is a neighbor of X_i, and these nodes must appear together in the elimination clique X_{T_i}.

3.2.3 Computational complexity

Let us now consider the computational complexity of ELIMINATE. At each step we must sum over a variable x_i for all configurations of the variables in the summand $\phi_i(x_{T_i})$. Assuming that there is no special algebraic structure in this summand that can be exploited, the time and space complexities are exponential in the number of variables in the subset T_i. That is, the overall complexity of the algorithm is determined by the number of variables in the largest elimination clique. Thus, the question of the computational complexity of ELIMINATE can be reduced to the purely graph-theoretic question of the size of the largest elimination clique created by UNDIRECTEDGRAPHELIMINATE.

The problem of obtaining a largest elimination clique that is as small as possible, under all possible elimination orderings, is a well-studied problem in graph theory. The problem is generally expressed in terms of a parameter k known as the *treewidth*, which is one less than the smallest achievable value of the cardinality of the largest elimination clique, where we range over all possible elimination orderings.

Consider for example, the star graph on n nodes shown in Figure 3.8(a). If we were to eliminate the central node first, we would immediately link all other nodes, creating an elimination clique of size n. On the other hand, if we eliminate all of the leaf nodes first we never create a clique of cardinality greater than two. Indeed, the treewidth of this graph is equal to one.

Figure 3.8(b) shows a second example, in which it can be verified that it is possible to eliminate the nodes in such a way that the largest clique created is of size three. The treewidth is thus equal to two.

The general problem of finding the best elimination ordering of a graph—an elimination ordering that achieves the treewidth—turns out to be NP-hard. We discuss this hardness result, and its consequences for probabilistic inference, in more detail in Chapter 17. Indeed, in that chapter we discuss an inference algorithm (the junction tree algorithm) that generalizes ELIMINATE and necessitates a deeper discussion of the treewidth problem and methods for tackling it. As we will show, there are a number of useful heuristics for finding good elimination orders, and these can provide viable solutions in practical problems.

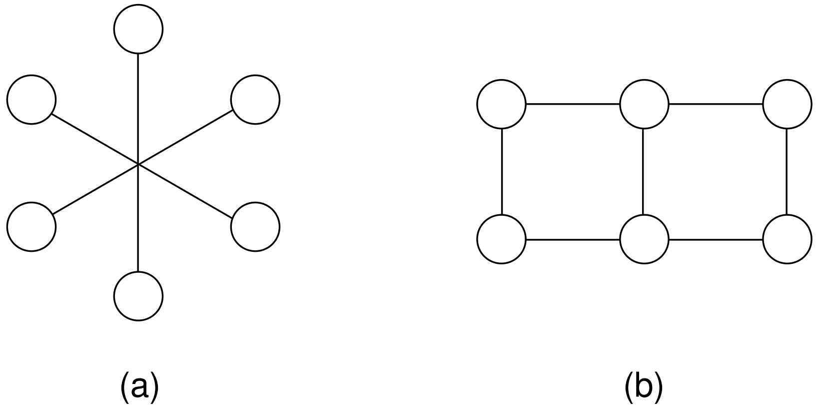

Figure 3.8: (a) A graph whose treewidth is equal to one. (b) A graph whose treewidth is equal to two.

In the meantime, all of the graphs that we study in intervening chapters will turn out to involve graphs that have "obvious" optimal elimination orderings.

The NP-hardness result should be taken as injecting a cautionary note into our study of elimination methods, suggesting that we should not expect ELIMINATE to provide a computationally-efficient solution to the general problem of probabilistic inference. On the other hand, we clearly should never have expected any such general solution from ELIMINATE. The fully-connected graph, for example, yields a single clique containing all of the nodes, under all possible elimination orderings, and thus has no graph-theoretic structure that ELIMINATE can exploit. To have any hope for efficient probabilistic inference in such a graph, we need to hope that other structural features of probability theory can be brought to bear.[5]

We can also take the NP-hardness result as providing a crisp statement of the computational bottleneck that arises in ELIMINATE. Indeed, note that UNDIRECTEDGRAPHELIMINATE provides a practically useful tool for assessing the severity of this bottleneck. For a given elimination ordering, we can obtain a cheap assessment of the predicted running time of ELIMINATE by running UNDIRECTEDGRAPHELIMINATE. If UNDIRECTEDGRAPHELIMINATE yields elimination cliques of reasonably small cardinality, then we know that it is viable to run ELIMINATE.

3.2.4 Graph elimination and directed graphs

All of the considerations of the previous three sections also apply to directed graphs. There is, however, a minor idiosyncracy of directed graphical models that must be addressed if we are to use the concept of "elimination clique" to analyze the directed version of ELIMINATE.

The functions that are used to initialize the active list in the directed case are conditional probabilities, $p(x_i \,|\, x_{\pi_i})$. Note that a pair of variables X_j and X_k that are parents of X_i are linked

[5]That is, there may be special algebraic structure in the potentials, or symmetries, or simplifications brought about by laws of large numbers. These issues will return in our consideration of approximate inference algorithms, in Chapter 20 and Chapter 21.

DIRECTEDGRAPHELIMINATE(G, I)
  G^m = MORALIZE(G)
  UNDIRECTEDGRAPHELIMINATE(G^m, I)

MORALIZE(G)
  for each node X_i in I
    connect all of the parents of X_i
  end
  drop the orientation of all edges
  return G

Figure 3.9: An algorithm for eliminating nodes in an directed graph G.

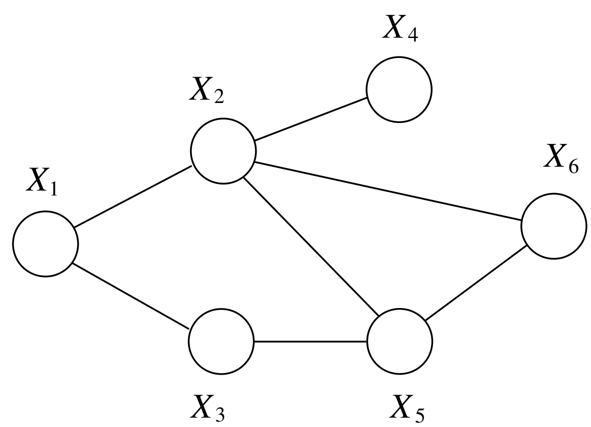

Figure 3.10: The moral graph corresponding to the directed graph in Figure 2.1.

functionally by their presence in the function $p(x_i \,|\, x_{\pi_i})$, but they are not necessarily neighbors in the graph $\mathcal{G}$ (e.g., X_2 and X_5 are not linked in Figure 3.1). This breaks the relationship between elimination cliques and sets of arguments that we established in the previous section for undirected graphs.

There is a simple fix. To define the elimination cliques for a directed graph, first connect all of the parents of each node—this captures the basic fact that factors involving all variables X_{π_i} will necessarily appear in our calculations. Then drop the orientation of all of the edges in the graph, converting the graph to an undirected graph. This procedure, of "marrying" the parents of the nodes in a directed graph and converting to an undirected graph, is referred to as *moralization*. The resulting graph is referred to as a *moral graph*. We use moralization as a subroutine in the algorithm, DIRECTEDGRAPHELIMINATE, for eliminating directed graphs (see Figure 3.9).

The graph in Figure 3.10 is the moral graph corresponding to the directed graph in Figure 2.1.

Note that (in the elimination order that we have been using in our example) X_6 is eliminated before its parents X_2 and X_5, and the elimination step already adds a link between these two nodes. That is, in this case, we do not need to moralize; elimination does it for us. On the other hand, if a parent of X_6 appears before X_6 in the elimination order, we need to moralize explicitly. Consider in particular an elimination ordering in which X_5 is eliminated first. The other parent, X_2, is not a neighbor of X_5 when the latter node is eliminated, and thus is not included within the elimination clique of X_5. This fails to capture the fact that summing over X_5 creates an intermediate factor that refers to X_2 and the other neighbors of X_5. In general we need to moralize in the directed graphical setting if we want the elimination cliques to capture all such dependencies.

The considerations in this section may help to explain the important role that undirected graphical models play in designing and analyzing inference algorithms, a role that we will see again in later chapters, even when the original graphical model is directed. In a directed model, the basic factors that appear in the joint probability are *conditional* probabilities. There is of course a great difference between the appearance of a variable on the left-hand or right-hand side of a conditional probability. From the point of view of ELIMINATE, however, this difference is irrelevant. When we sum over x_k, the factor $p(x_i \mid x_j, x_k)$ and the factor $p(x_j \mid x_i, x_k)$ both create an intermediate factor linking x_i and x_j. Thus, to understand the computational complexity of inference, we need to ignore the directionality associated with a conditional probability. The undirected graphical formalism, which treats such a probability as a general potential, $\psi(x_i, x_j, x_k)$, does this automatically.

3.3 Discussion

Our presentation of the elimination algorithm for probabilistic inference raises a number of questions:

- Can we prove that it works?
- What are its limitations?
- Can it be generalized?

Detailed answers to these questions will emerge in later chapters, but let us try to provide some short answers here.

It is not difficult to prove that the algorithm that we have presented is correct. Indeed, we ask the reader to provide a proof by induction in Exercise **??**, and we present a proof by induction of the correctness of a closely-related algorithm (the SUM-PRODUCT algorithm) in Chapter 4. Moreover, in Chapter 17 we prove the correctness of the general junction tree algorithm, an algorithm that generalizes both ELIMINATE and SUM-PRODUCT.

The ELIMINATE algorithm has a number of limitations, some which are easily corrected and others which are not. In particular, taking ELIMINATE seriously as an algorithm to be implemented on a computer reveals a number of inefficiencies. Most importantly, the use of a single "active list" as a data structure requires an inefficient traversal of the entire list every time the algorithm eliminates a node. This can be fixed by maintaining a separate list, or "bucket," for each node. Whenever the algorithm creates a new intermediate factor, $m_i(x_{S_i})$, it scans the elimination ordering I, and

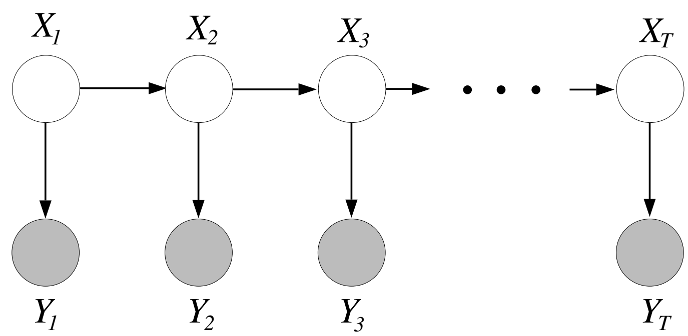

Figure 3.11: A chain-structured graphical model.

finds the first occurrence of an index in S_i. It then places $m_i(x_{S_i})$ in the bucket associated with that index. This "bucket elimination" approach to elimination is explored in Exercise **??**.

A more serious limitation of the basic elimination methodology is the restriction to a single query node. While it is not difficult to develop variations of the algorithm that handle small sets of interconnected query nodes, significantly more work is required to generalize the algorithm further, in particular to handle the common situation in which we require the conditional probability of *all* of the non-evidence nodes in the graph.

Consider, for example, the graphical model shown in Figure 3.11, an important graph that we will meet again in Chapter 12 and Chapter 15. Here we have a backbone of unshaded nodes X_i for which we require the conditional probabilities, where we condition on the shaded nodes Y_i hanging off the backbone. Computing the conditional probability of any single node is a straightforward application of ELIMINATE. Thus, for example, we can calculate $p(x_1 \mid y)$, where $y = \{y_1, y_2, \ldots, y_n\}$, by defining an elimination ordering in which x_1 is the final node. Similarly we can choose any intermediate node x_i as the final node in an elimination ordering; for example, we can choose an elimination ordering in which the flow is forward from x_1 to x_i, and backward from x_n to x_i.

We can obviously calculate the conditionals by running the elimination algorithm n times, once for node x_i. Clearly, however, this approach is inefficient, requiring us to repeat the same elimination steps many times. For example, in calculating $p(x_1 \mid y)$ and $p(x_2 \mid y)$, all of the steps involved in marginalizing over $x_3, \ldots, x_n$ would be repeated.

It is not difficult to figure out how to avoid the redundant calculations in the case of the graph in Figure 3.11, and indeed we will present various algorithms in Chapter 12 and Chapter 15 that calculate all of the desired conditionals via a single forward and backward pass along the chain. What we would like, however, is a general procedure for avoiding redundant computation.

We will make the step up to such a general procedure in two steps. First, in Chapter 4, we describe the SUM-PRODUCT algorithm, a procedure that allows the computation of all singleton marginals, but is restricted to trees. Second, in Chapter 17, we put together what we have learned from the elimination approach and the sum-product algorithm, and describe a general procedure—the junction tree algorithm—that computes marginals for general graphs. The key idea behind the junction tree algorithm is to avoid the multiple, different elimination orderings that repeated runs of elimination would require, in essence by developing a general data structure for caching and combining intermediate factors. Rather than focusing on elimination orderings, the junction tree algorithm focuses on the relationships between intermediate factors, or "messages." The same idea is present in simpler form in the SUM-PRODUCT algorithm, to which we now turn.

3.4 Historical remarks and bibliography