An Introduction to Probabilistic Graphical Models

Michael I. Jordan
University of California, Berkeley

June 30, 2003

Chapter 17

The Junction Tree Algorithm

In earlier chapters we have presented a number of examples of inferential calculations in graphical models. The general problem has been to calculate the conditional probability of a node or a set of nodes, given the observed values of another set of nodes. In the case of mixture models and factor analysis models the problem was to calculate the conditional probabilities of the latent variables given the observed data, and the solution was a rather straightforward application of Bayes rule. In the case of the HMM and the state-space model we saw a somewhat more complex inference problem involving dependencies between nodes arranged in a sequence. The solution was again an application of Bayes rule, but it was necessary to find recursions that allowed the inference problem to be solved efficiently. The Markov properties of the underlying graphical model provided the formal machinery to justify these recursions.

In the current chapter we present a general approach to inference that makes systematic use of the Markov properties of graphical models. All of the examples that we have treated until now emerge as special cases; moreover, the recursions that we worked out rather painstakingly in each individual case can now be derived more systematically. The general idea is to use the Markov properties of graphical models to find ways to decompose a general probabilistic calculation into a linked set of local computations. The key to this approach is an appropriate definition of "local."

Chapter 3 presented a simple elimination algorithm (ELIMINATION) for inference on directed or undirected graphs. As ELIMINATION runs it creates dependencies between nodes, in effect redefining the "locality" relationships in the graph. To develop a deeper understanding of probabilistic inference, it proves helpful to abstract away from the specific process of elimination and to focus on this general notion of locality. In effect we shift our focus from the *process* of inference to the *data structures* that underly inference. We find that a particular data structure—the *junction tree*—emerges from these considerations. The junction tree makes explicit the important (and beautiful) relationship between graph-theoretic locality and efficient probabilistic inference.

Although we present specific algorithms for probabilistic inference in this chapter, it is important to emphasize at the outset that our goal is less that of providing specific recipes as it is of providing an understanding of the key general concepts that underly inference. Thus, while we will describe concrete algorithms (the "Hugin algorithm," the "Shafer-Shenoy algorithm," and the "Lauritzen-Spiegelhalter algorithm"), we view all of these algorithms as instances of a general algorithmic framework that we will refer to generically as the *junction tree algorithm.* Understanding the general framework makes it easy to see how various specific algorithms arise and how they interrelate. Moreover, an important bonus of developing the general junction tree framework is the realization that probabilistic inference is itself an instance of a more general class of problems, all of which involve factorized potentials on graphs, and all of which can be solved using suitable variations on the junction tree theme. We discuss some instances of this more general class at the end of the chapter.

We begin by returning to the elimination algorithm from Chapter 3, stripping away some of its inessential details, and aiming to overcome some of its deficiencies.

17.1 From elimination to the junction tree

In Figure 17.1(a) we show the graph that served as a running example in Chapter 3. The factored form of the joint probability distribution for this graph is as follows:

$$p(x_1, x_2, \ldots, x_6) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3)p(x_6 \mid x_2, x_5). \tag{17.1}$$

As in Chapter 3 we will use the elimination ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$ in our examples.

Each factor in Eq. (17.1) expresses a dependency among one or more variables. Forming summands during a run of the elimination algorithm creates additional dependencies—for example, summing over x_6 creates an intermediate factor that is a function of x_2 and x_5. The *elimination cliques* associated with an elimination ordering can be viewed as an explicit record of these dependencies. Recall that we can abstract away from probabilistic inference and view these elimination cliques as being formed by a purely graph-theoretic procedure (called UNDIRECTEDGRAPHELIMINATE in Chapter 3) in which we link all of the neighbors of a given node (thus forming a clique), and remove the node from the graph. In particular, for the elimination ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$, the elimination cliques are as shown in Figure **??**(b). While the elimination algorithm ELIMINATION does not explicitly form these cliques, the graph-theoretic operation of forming elimination cliques parallels the algebraic operation of marginalizing over a node, and neatly summarizes the graphical consequences of marginalization.

The elimination algorithm is "query-oriented." That is, the algorithm yields the marginal or conditional probability of a given query node—the last node in the elimination ordering. Intermediate factors that are created along the way are discarded. While in some cases this is what we want, in many cases it is not. Consider in particular the chain-structured graphical model associated with the HMM or the state-space model. To calculate the posterior probability of any particular node we can eliminate foreward and backward until we arrive at the node. In doing so we create a number of intermediate factors. Many of these same intermediate factors can be used in calculating the posterior probability of other nodes. Clearly we wish to avoid recomputing such factors, as we would do in a naive application of elimination. We also need to know which intermediate factors are needed for which posterior probabilities and how to combine factors—in essence we need a calculus for the intermediate factors. The elimination algorithm provides us with little help in this regard.

As a first step in moving beyond the elimination algorithm we need to allocate data structures—"permanent storage"—to the intermediate factors. Each such factor is associated with one of the

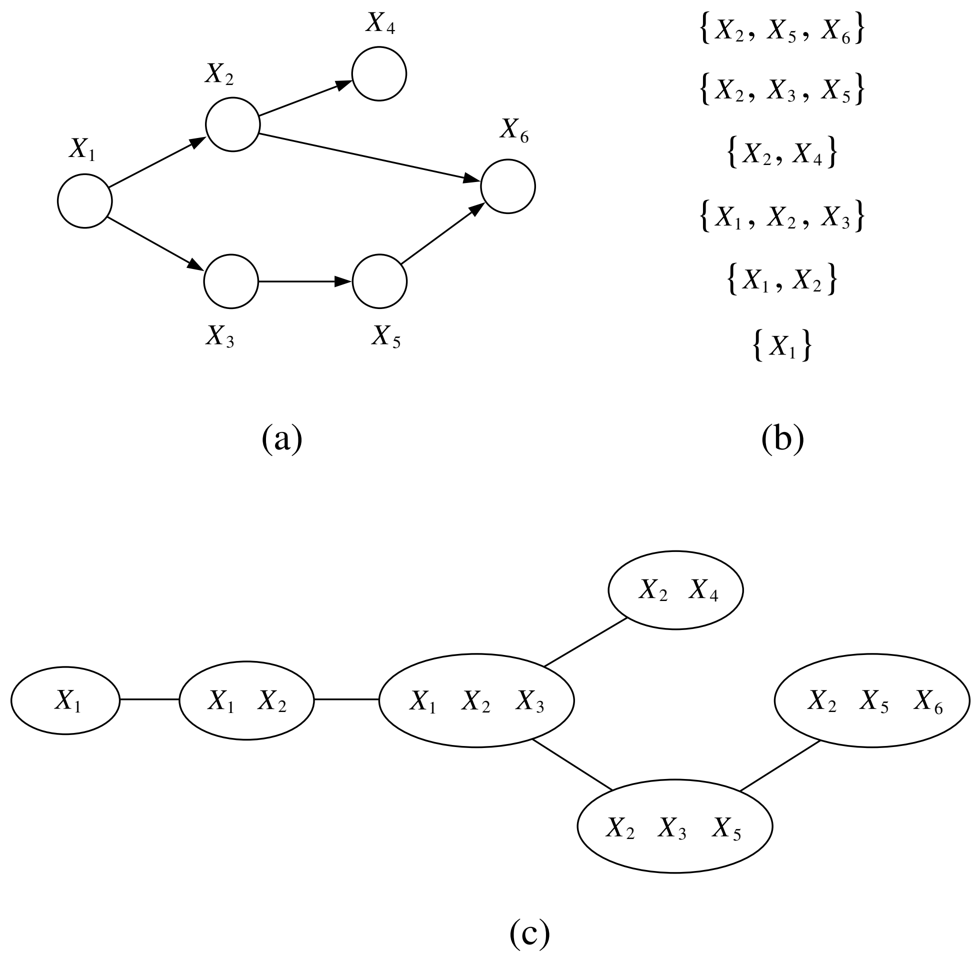

Figure 17.1: (a) The six-node example from Chapter 3. (b) The elimination clique created from a run of the elimination algorithm using the ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$. (c) The elimination cliques arranged into a clique tree.

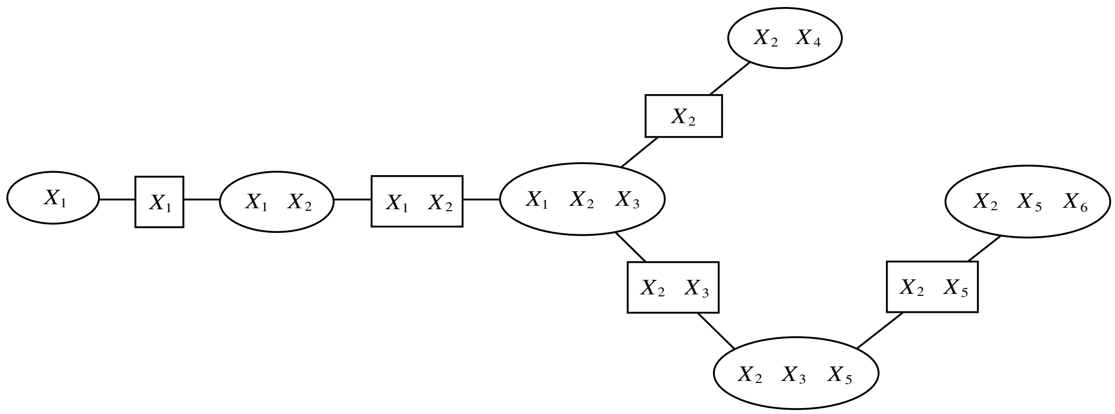

Figure 17.2: A clique tree annotated with separator sets.

elimination cliques in Figure 17.1(b). We can therefore view the nodes in this figure as representations of the storage that we need if we are to record the intermediate factors created during a run of the elimination algorithm.

While a list of the elimination cliques reveals some of the structure associated with the elimination algorithm, there is additional structure that is worth noting. In particular, as we have seen, summing over a variable produces an intermediate factor that subsequently appears in the summand associated with a later variable. For example, summing over x_5 creates an intermediate factor that refers to x_3 and thus appears in the summand when we subsequently sum over x_3. If we view the nodes in Figure 17.1(b) as storage sites, and if we view the operation of summing as operating on the data stored at these sites, then it is natural to try to represent the transfer of information between these sites. For example, the sum over x_3 requires the factor created at the x_5 site, and we therefore need to transfer this factor between the site corresponding to the elimination of x_5—the elimination clique $\{X_2, X_3, X_5\}$—and the site corresponding to the elimination of x_3—the elimination clique $\{X_1, X_2, X_3\}$. As shown in Figure 17.1(c), we can capture this flow of information by drawing an edge between these elimination cliques.

The graphical object in Figure 2.1(c) is a *clique tree*—a singly-connected graph in which the nodes are the cliques of an underlying graph. Every run of the elimination algorithm can be viewed as implicitly creating a clique tree—the clique tree can be viewed in essence as an "execution trace" of the algorithm. What we are groping towards, however, is an algorithm that goes beyond the elimination framework by explicitly representing a clique tree as a data structure. The nodes in such a clique tree will store intermediate factors, allowing these factors to be reused in multiple queries. Information will flow around the clique tree in multiple directions.

In Figure 17.2 we annotate the clique tree with some additional structure that will prove to be useful. Between each linked pair of cliques we introduce a *separator set*—the intersection of the corresponding cliques. The separator sets are themselves cliques, being the intersection of cliques. These sets provide an explicit representation of the variables referred to by the intermediate factors that pass between cliques. Consider, for example, the intermediate factor created at the clique $\{X_2, X_3, X_5\}$. Summing over x_5 creates a factor that is a function of x_2 and x_3, and this factor

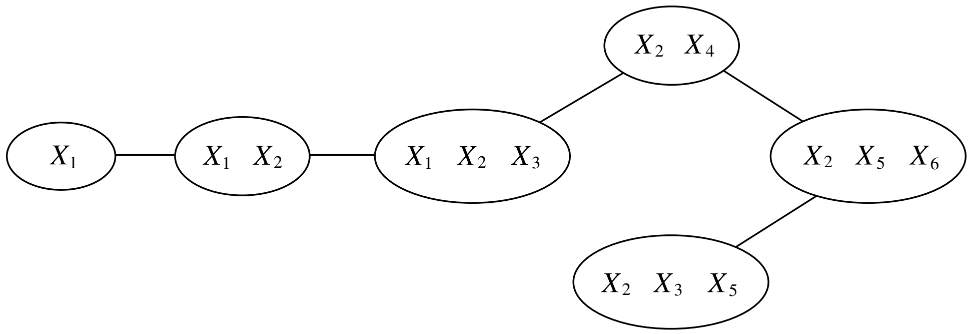

Figure 17.3: A clique tree that does not possess the junction tree property. Note in particular that the cliques containing the node X_3 do not form a connected subtree.

is sent to the clique $\{X_1, X_2, X_3\}$, where we subsequently sum over x_3. The separator set on the link between these cliques contains the nodes $\{X_2, X_3\}$, and thus explicitly represents the domain of the intermediate factor transferred between the cliques.

Not all clique trees are created equal. In particular, the clique tree in Figure 2.1(c) has some special properties. Note that the index "2" appears in five different nodes in the figure, and that these five nodes are connected—they form a *connected subtree*. Moreover, this is true of all of the other node indices. This interesting and important property is known as the *junction tree property*. Not all clique trees possess the junction tree property; for example, the tree in Figure 17.3 does not possess the junction tree property. As we will see in the remainder of the chapter, understanding the junction tree property is the key to a general understanding of probabilistic inference.

17.2 Potentials

With the discussion in the previous section as background, we embark on a general discussion of the junction tree algorithm. We will be focusing on a particular variant of the general junction tree algorithm known as the "Hugin algorithm," and will discuss other variations in later sections and in the exercises.

Let $G = (V, E)$ denote a directed or undirected graph with vertices V and edges E. Let $\mathcal{C}$ denote a set of *cliques* of G; i.e., $\mathcal{C}$ is a set of completely connected subsets of V. We generally require these subsets to be maximal, so that no member of $\mathcal{C}$ is a subset of another member of $\mathcal{C}$. However, at the cost of a bit of redundancy it is at times convenient to allow such proper subsets to appear in $\mathcal{C}$.

Let X be a random vector indexed by the vertices V. Recall that we allow subsets of the vertex set V to be used as indices; thus, corresponding to each clique $C \in \mathcal{C}$, we have a set of random variables X_C, with realizations x_C. The number of such realizations is the product of the number of realizations of each individual random variable X_u, for $u \in C$.

Associated with each $C \in \mathcal{C}$ we define a *potential* $\psi_C(x_C)$, a nonnegative function on the realizations x_C. In general there are no constraints on the potential functions other than nonnegativity. Note in particular that the sets C can overlap, and we make no "consistency" requirements on the

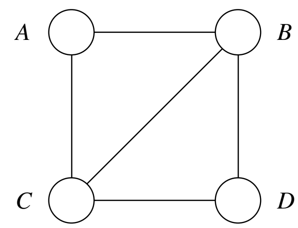

Figure 17.4: A four-node model which we assume is parameterized with pairwise potentials: ψ_{AB}, ψ_{AC}, ψ_{BC}, ψ_{BD}, and ψ_{CD}.

overlap.

We now define a joint probability distribution on X as the normalized product of potential functions:

$$p(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C). \tag{17.2}$$

This is of course the same definition as that used for undirected graphs. Note, however, a subtle but important change in focus—in the current section we view the set of subsets $\mathcal{C}$ as an explicit data structure, with the underlying graph in the background. Technically, our data structure is a *hypergraph*—a set of subsets—with Eq. (17.2) defining the joint probability distribution associated with the hypergraph.

There are problems in which it is natural to pose the problem directly in terms of factored potentials on sets of subsets, without focusing on an underlying graph. Most commonly, however, the potentials on the hypergraph are *initialized* from those of an underlying graph. Let us consider how this initialization process works for both undirected and directed graphs.

Undirected graphs come endowed with potential functions on cliques, and if these cliques are the same as the set of subsets $\mathcal{C}$, then the initialization problem is vacuous; we simply define $\psi_C(x_C)$ to be the corresponding potential from the underlying graph. In general, however, these sets are not the same. In particular, we generally include only the maximal cliques in the set $\mathcal{C}$. If the parameterization of the underlying undirected graph is restricted to cliques that are proper subsets of the maximal cliques of the graph, as is often the case, then we have a many-to-one mapping from parameterized cliques to $\mathcal{C}$. Consider, for example, the undirected graphical model in Figure 17.4, where we assume that the model is parameterized via pairwise potentials. The maximal cliques of the graph are, however, triplets of nodes. In such a situation, the potentials on maximal cliques in Eq. (17.2) are formed as the product of potentials from the underlying graph. Thus, in our example, we define ψ_{ABC} to be the product $\psi_{AB}\psi_{AC}$, while we define ψ_{BCD} to be the product $\psi_{BC}\psi_{BD}\psi_{CD}$. Note that ψ_{BC} can be associated with either triple; we have arbitrarily assigned it to ψ_{BCD}. In general each potential ψ_D on the underlying graph is assigned to one and only one ψ_C on the hypergraph, where $D \subset C$. If we assume that $\mathcal{C}$ includes the maximal cliques, then this can always be done.

Having assigned each underlying potential to one and only one ψ_C, the product in Eq. (17.2) is a faithful representation of the joint probability from the underlying graph.

Similar issues arise when we initialize a set of clique potentials from an underlying directed

```
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 17.5: An algorithm to moralize a directed graph.

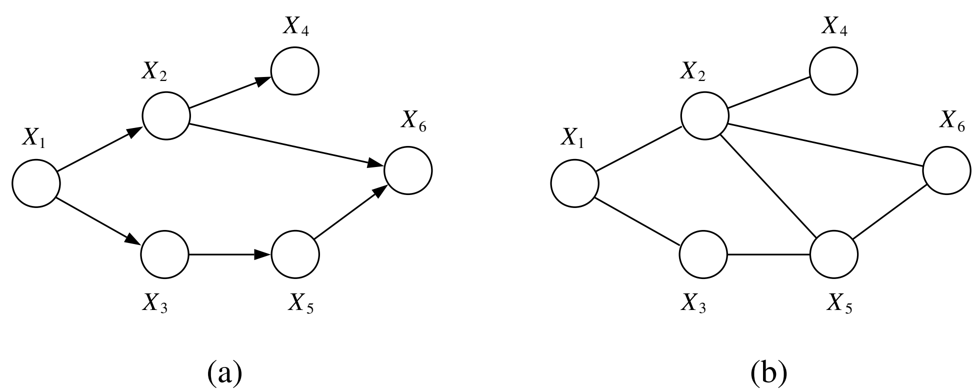

Figure 17.6: (a) A directed graph. Note that the conditional probability $p(x_6 \,|\, x_2, x_5)$ has as arguments a subset of nodes that are not contained in any clique in the graph. In the moral graph in (b), an edge has been added between X_2 and X_5, and now the arguments in the potential $p(x_6 \,|\, x_2, x_5)$ are contained with the clique $\{X_2, X_5, X_6\}$.

graph, with the additional complication that the original potentials—the local conditional probabilities from the directed graph—need not be defined on cliques. In particular, if the parents of node X_i are not linked, then $p(x_i \,|\, x_{\pi_i})$ is not a function on a clique. To handle this situation, and thereby allow a uniform treatment of directed and undirected graphs, we *moralize* the directed graph. Recall from Chapter 3 that the moral graph G^m corresponding to a directed graph G is obtained by linking the parents of each node and dropping the directionality of the edges. We define the moralization procedure more formally in Figure 17.2. On a moral graph, the local conditional probabilities are potential functions on cliques. We associate each such probability with one and only one potential $\psi_C(x_C)$, again assuming that $\mathcal{C}$ includes the maximal cliques. Taking the product over these potentials is then equivalent to taking the product $\prod_i p(x_i \,|\, x_{\pi_i})$, and faithfully represents the joint probability from the underlying directed graph.

Note that for directed graphs the potentials are already normalized; in other words, the normalization factor Z is automatically one.

Figure 17.2 shows an example for a directed graph.

Note that the moralization procedure adds edges to a directed graph. How does this procedure square with the semantic distinctions between directed graphs and undirected graphs presented in the previous chapter? Recall that a given graph—directed or undirected—is associated with a family of probability distributions. This family can be specified by writing down the list of conditional independence statements associated with the graph. Any distribution that respects all of the conditional independence statements in the list belongs to the family. Clearly, if we make *fewer* statements we make the family *larger*. Now note that a moral graph necessarily makes fewer conditional independence statements than its corresponding directed graph. In particular, a directed graph asserts all of the conditional independencies that characterize the moral graph, as well as additional independencies between the parents of a given node in the marginal distribution in which the node is eliminated. Thus the set of probability distributions associated with the directed graph is a subset of the set of probability distributions associated with the moral graph. If we solve the inference problem for the family of probability distributions associated with the undirected moral graph, we solve it for the family of probability distributions associated with the directed graph as well.

Moralization is not merely a convenience, but is also a necessary component of any inference algorithm. Marginalization or conditioning couples the parents of a node, creating an intermediate factor that is in general a non-trivial function of the parents.[1] Intuitively, moralization is necessary to capture dependencies such as "explaining-away" that arise whenever a node is an evidence node or has descendants that are evidence nodes.

To summarize, our procedure will be to identify the maximal cliques of an undirected or (moralized) directed graph.[2] We initialize the potential functions associated with these cliques from the potentials and local conditional probabilities on the underlying graph.

17.3 Introducing evidence

We now consider the problem of conditioning, or "introducing evidence." We suppose that the nodes are partitioned into subsets H and E, and that the random vector X_E is observed to take on a specific value. The problem that we discuss in this section is that of representing the conditional probability $p(x_H \mid x_E)$. Once we have decided on such a representation, the inferential problem of computing marginals under this probability—the conditional probabilities of subsets of the nodes X_H—will be no different in principle from the calculation of marginal probabilities under the overall joint $p(x)$.

Our general approach will be to represent conditionals via taking "slices" of the potentials defining the joint probability. Suppose in particular that we have represented the joint probability as a product over cliques as in Eq. (17.2). For each clique C, consider the intersection $C \cap E$. The nodes in this intersection have been fixed to specific values, and the potential in effect now ranges

[1] If a node is not an evidence node or has no descendants that are evidence nodes, summing over the values of the node yields the trivial value of one.

[2] Some readers may wonder how we can achieve this—finding maximal cliques is an NP-hard problem! In fact, we will not be finding the maximal cliques of arbitrary graphs, but only of a special class—the *triangulated graphs*. Maximal cliques of triangulated graphs can be found easily. Let us postpone our discussion of triangulation, however, at the cost of a bit of naiveté with regards to identifying maximal cliques.

over the complement (in C) of this set of nodes, i.e., $C \cap H$. where $C = (C \cap H) \cup (C \cap E)$ by the assumption that H and E partition V. Thus, for a particular fixed configuration $\bar{x}_E$, we have:

$$p(x_H, \bar{x}_E) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_{C\cap H}, \bar{x}_{C\cap E}). \tag{17.3}$$

This is a product of "slices" of potential functions.

A slice of a potential function is itself a potential function. Thus we can also view Eq. (17.3) as a product of potential functions on subsets $\{X_{C\cap H}$ of the nodes X_H, suppressing reference to the nodes X_E. That is, writing $\tilde{\psi}_{C\cap H}(x_{C\cap H}) \triangleq \psi_C(x_{C\cap H}, \bar{x}_{C\cap E})$ to suppress the explicit reference to the fixed configuration $\bar{x}_E$, we have:

$$p(x_H, \bar{x}_E) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \tilde{\psi}_{C\cap H}(x_{C\cap H}) \tag{17.4}$$

as a product of potential functions over X_H.

There is an oddity to Eq. (17.4), however, in that the normalization factor Z is obtained by summing over both X_H and X_E, whereas the product is defined only over X_H. It should be no surprise that Z is not in fact the normalization factor for the product of potentials $\tilde{\psi}_{C\cap H}$; indeed, this product is not normalized. Let us compute the normalization factor. Summing over H, and denoting the sum as $\tilde{Z}$, we compute:

$$\begin{aligned} \tilde{Z} &\triangleq \sum_H p(x_H, \bar{x}_E) && (17.5)\\ &= \sum_H \frac{1}{Z} \prod_{C \in \mathcal{C}} \tilde{\psi}_{C\cap H}(x_{C\cap H}). && (17.6) \end{aligned}$$

We also know, however, that $\sum_H p(x_H, \bar{x}_E) = p(\bar{x}_E)$, by definition. Putting these facts together, we have:

$$\frac{p(x_H, \bar{x}_E)}{p(\bar{x}_E)} = \frac{\prod_{C\in\mathcal{C}} \tilde{\psi}_{C\cap H}(x_{C\cap H})}{\sum_H \prod_{C\in\mathcal{C}} \tilde{\psi}_{C\cap H}(x_{C\cap H})}. \tag{17.7}$$

That is, the slices $\tilde{\psi}_{C\cap H}(x_{C\cap H})$ provide a potential function representation of the *conditional* probability $p(x_H \,|\, \bar{x}_E)$. The normalization factor for this representation is the marginal probability $\tilde{Z} = p(\bar{x}_E)$. Note that the original normalization constant, Z, cancels when we form the ratio on the right-hand-side of Eq. (17.7). Thus, for the purpose of calculating conditional probabilities, we have no need of knowing the normalization constant associated with the original set of potentials; it suffices to compute the normalization constant of the sliced potentials.

Let us see how this works for a particularly simple case. In Figure 17.7. we show a directed graph and the corresponding moralized graph for two binary nodes X and Y. Given the three probabilities $p(X=1) = .8$, $p(Y=1 \,|\, X=1) = .7$ and $p(Y=1\,|\,X=0) = .4$, we can construct a joint probability distribution. Converting to a set of cliques, we have a single clique $\{X, Y\}$, with clique potential given by the product $p(x)p(y\,|\,x)$:

$$\psi_{\{X,Y\}} = \begin{bmatrix} .12 & .08 \\ .24 & .56 \end{bmatrix} \tag{17.8}$$

Figure 17.7: A two-node graphical model with its moralized graph.

Given that this potential arises from a directed graph, it is no surprise that the clique potential is normalized. Suppose that we now observe evidence $Y = 1$. We obtain the slice:

$$\tilde{\psi}_{\{X\}} = \begin{bmatrix} .08 \\ .56 \end{bmatrix}, \tag{17.9}$$

which is a function only of X. Note that this new clique potential is unnormalized. Normalizing yields the number $\tilde{Z} = .64$, which we recognize as the probability $p(Y = 1)$. Morever, the normalized potential is given by dividing $\tilde{\psi}_{\{X\}}$ by $\tilde{Z} = .64$:

$$\frac{1}{\tilde{Z}}\tilde{\psi}_{\{X\}} = \begin{bmatrix} .125 \\ .875 \end{bmatrix}, \tag{17.10}$$

which is the conditional distribution $p(x \mid Y = 1)$.

To summarize, our general representation of a probability distribution is a (possibly) unnormalized set of potentials on a set of cliques. Conditioning is handled by restricting attention to subsets of the original set of cliques, and by defining potentials on these subsets that are slices of the original potentials. In general we make no fundamental representational distinction between conditional and joint distributions.

This perspective also helps to reveal more of the unity in undirected and directed representations of probabilities. In the directed case, the set of potentials is normalized at the outset: $Z = 1$. But as soon as we observe evidence, the resulting set of slices is no longer normalized, and the conditional distribution is represented as an unnormalized product of potential functions, as in the undirected case.

An equivalent approach to representing conditional probability distributions involves introducing "evidence potentials." An evidence potential is a delta function, $\delta(x_E, \bar{x}_E)$, i.e., a function which is equal to one if its arguments are equal and zero otherwise. We used evidence potentials in our presentation of the elimination algorithm in Chapter 3. Multiplying the original product of potentials by the evidence potential yields an unnormalized product on the set (X_H, X_E). Summing over x_E has the effect of setting $p(x_H, x_E)$ equal to $p(x_H, \bar{x}_E)$. Thus we obtain the same representation as that considered in this section, once we "marginalize" and restrict attention to X_H. The approach based on evidence potentials is elegant because it treats slices as formally equivalent to marginalization; indeed that was the reason that we introduced it in Chapter 3. In practice, however, using evidence potentials involves introducing zeros and then summing over those zeros. As an algorithmic matter it is more efficient to simply take slices.

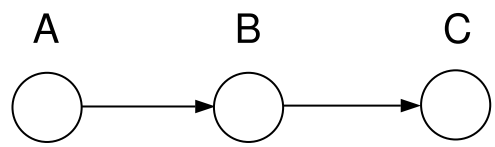

Figure 17.8: A three-node Markov chain.

17.4 Clique trees

We now begin to address the crux of the problem, which is that of computing *marginal probabilities*. Thus, we wish to compute the marginal $p(x_F \,|\, x_E)$, where (F, G) is a partition of H and where F ranges over a set of subsets of interest. In particular, we may wish to compute all probabilities $p(x_F \,|\, x_E)$, where F ranges over all singleton nodes. More generally, we will address the problem of computing $p(x_F \,|\, x_E)$, where F ranges over all cliques in $\mathcal{C}$, and over all subsets of these cliques.

A still more general problem is that of computing $p(x_F \,|\, x_E)$ for arbitrary F, and while we will address this problem in Section **??**, it is worth noting that in most applications it suffices to compute marginal probabilities for the cliques. In particular, the cliques are sufficient statistics for distributions that factor according to Eq. (17.2); thus, for computing expected sufficient statistics in the context of an EM algorithm it suffices to obtain clique marginals.

We define a *clique tree* as a singly-connected graph whose nodes represent members of the clique set $\mathcal{C}$. Edges in this graph will allow us to define information flows between cliques. The junction tree algorithm can be understood as an algorithm that uses these information flows to manipulate the clique potentials so as to yield marginal probabilities. In particular, after the algorithm runs, the potential ψ_C will be equal to the marginal probability $p(x_C, \bar{x}_E)$. This probability is an unnormalized version of the conditional $p(x_C \,|\, \bar{x}_E)$, where the normalization constant is obtained by summing or integrating ψ_C over x_C. Thus, we can obtain the desired marginal probabilities via a local operation. The goal of the remainder of the chapter is to explain how this is achieved.

In the previous two sections, we showed how to initialize the clique potentials so as obtain a representation of the joint or conditional probability. This is a global representation; the individual potentials do not necessarily correspond to local probabilities. Consider in particular the Markov chain shown in Figure 17.8. The cliques of this graph are $\{A, B\}$ and $\{B, C\}$. The joint probability is $p(x_A, x_B, x_C) = p(x_A)p(x_B \,|\, x_A)p(x_C \,|\, x_B)$, and while $p(x_A)$ and $p(x_B \,|\, x_A)$ can be grouped to initialize the potential ψ_{AB} to the marginal $p(x_A, x_B)$, the remaining factor $\psi_{BC} = p(x_C \,|\, x_B)$ is not a marginal. To convert this potential into a marginal, we marginalize ψ_{AB} to obtain $p(x_B)$, and multiply ψ_{BC} by this factor. The transfer of the probability $p(x_B)$ is an instance of the information flow that we referred to above.

After adjusting ψ_{BC} we have achieved the goal of obtaining marginal probabilities for both of the cliques, but we have also lost something. In particular, the joint probability on (x_A, x_B, x_C) is not equal to the product of marginals $p(A, B)$ and $p(B, C)$, and thus the product of the clique potentials is no longer a representation of the joint probability.

The junction tree approach in essence allows us to have our cake and eat it too, retaining a representation of the joint probability while also manipulating the clique potentials so as to convert them into marginal probabilities. This is done by utilizing an extended representation of joint probabilities that makes use of the separator sets discussed in Section **??**. The remainder of this section introduces this important generalized representation.

On each edge of a clique tree we associate a *separator set* which contains the intersection of the cliques that it links. For example, in Figure 17.16, the separator is the singleton X_B. For a general clique tree on N nodes, we have $N-1$ separators.

We now augment our potential-based representation of joint probabilities to include potential functions on the separators as well as the cliques. Thus, letting $\mathcal{S}$ denote the set of all separators, we introduce a potential function $\phi_S(x_S)$ for each $S \in \mathcal{S}$. Given a clique tree with cliques $\mathcal{C}$ and separators $\mathcal{S}$ we define the joint probability as follows:

$$p(x) = \frac{\prod_C \psi_C(x_C)}{\prod_S \phi_S(x_S)}. \tag{17.11}$$

Note that we have omitted explicit reference to a normalizing constant Z. We adopt a convention of including the empty set as one of the separators and letting the "potential" on this empty set be the normalizing constant Z.

We have several questions to answer regarding this extended representation, but let us first return to our example and show what the representation achieves for us.

Expanding the joint probability associated with Figure 17.8, we have:

$$\begin{aligned} p(x_A, x_B, x_C) &= p(x_A, x_B) p(x_C \mid x_B) & (17.12) \\ &= \frac{p(x_A, x_B) p(x_B, x_C)}{p(x_B)}. & (17.13) \end{aligned}$$

This has the form of the extended representation shown in Eq. (17.11), where we define $\psi_{AB} = p(x_A, x_B)$, $\psi_{BC} = p(x_B, x_C)$, and $\phi_B = p(x_B)$. Thus, making use of the flexibility offered by the separator potentials, we are able to achieve a representation that is a product of marginals, and yet is also a representation of the joint probability. It turns out that we can always find this kind of representation for a given probability distribution. The proof of this fact will emerge during our development of the junction tree algorithm.

In our discussion of the Hammersley-Clifford theorem in Chapter 16, we showed that the representation of joint probability in Eq. (17.2) is general, in the sense that it allows us to capture all of the joint probability distributions that respect the conditional independence statements asserted by a graph. Clearly the extended representation includes all such joint probability distributions (set the separator potentials to unity). Does it include any others? The answer is no. This is seen by noting that the separators are (by definition) subsets of one or more cliques. Associating each separator with one such clique, and dividing that clique potential by the separator potential, we obtain a new set of clique potentials that represent the same joint, but without the separators. Thus the separator potentials do not enlargen the set of joint probability distributions that we can represent. They are essentially a convenience—they allow us to represent the set of joint probability distributions associated with a graphical model in a more flexible way.

An additional issue that we need to consider is the possibility of division by zero. We allow division by zero but only in a constrained set of circumstances. In particular, we define a separator

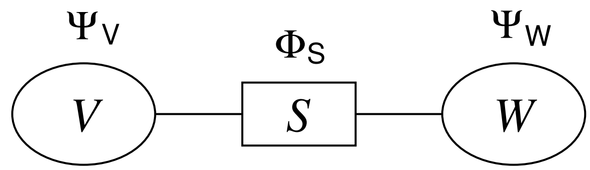

Figure 17.9: The basic data structures underlying the flow of information between cliques V and W.

potential to be *supportive* if whenever a configuration yields a value of zero for the separator potential, the clique potentials at both ends of the edge containing that separator also evaluate to zero. Thus we can never divide by zero in Eq. (17.11) unless the numerator is also zero. In this case we define the ratio to be zero. This makes sense—if a clique potential is zero for a configuration then the probability of that configuration should also be zero.

Each step of the junction tree algorithm is guaranteed to maintain supportiveness (see Exercise **??**). Thus, if we have supportive separator potentials at the outset then we maintain supportiveness as the algorithm runs.

We initialize the separator potentials to unity. Thus, at the outset, once we have introduced evidence, the set of clique potentials and separator potentials are (as before) a global representation of the joint conditional probability $p(x_H \,|\, x_E)$. The new capability that the extended representation has provided is the ability (in principle) to obtain a local representation of marginal probabilities, while maintaining an overall representation of the joint. We now show how this is achieved in practice.

17.5 Local consistency

Note that cliques can overlap, so the same node can appear in multiple cliques. Clearly, if the potentials are to represent marginal probabilities, it is necessary that they be consistent with each other; that is, they must give the same marginals for nodes that they have in common. This seemingly innocuous observation is the germ of the junction tree algorithm. We will find that consistency is not only a necessary condition, but it is also a sufficient condition for a probabilistic inference algorithm. Moreover, it turns out not to be necessary to compare all pairs of cliques that intersect; it will suffice to arrange the cliques into a special clique tree—a "junction tree"—and require only that cliques that are neighbors in the junction tree agree on the nodes that they have in common.

Let us postpone the general junction tree construction, and instead focus on the elemental problem of achieving consistency between a pair of cliques. Suppose that we have two cliques V and W and suppose that V and W have a non-empty intersection S (see Figure 17.9). The cliques V and W have potentials ψ_V and ψ_W, and we also endow S with a potential ϕ_S that we initialize to unity. The basic operation of the junction tree algorithm is an exchange of information between V and W, with S serving as a conduit for the flow of information. We first update W based on V, where the asterisk means "updated value of":

$$\phi_S^* = \sum_{V \setminus S} \psi_V \tag{17.14}$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W. \tag{17.15}$$

The first equation *marginalizes* the potential ψ_V with respect to S, storing the result in the separator potential. The second equation *rescales* the potential on W by multiplying by an "update factor" that is the ratio of the new separator potential to its old value.

This update has an important invariant: the joint distribution $p(x_H, \bar{x}_E)$. Note that ψ_V is unchanged during the update. Defining $\psi_V^* = \psi_V$, we have:

$$\frac{\psi_V^* \psi_W^*}{\phi_S^*} = \frac{\psi_V \psi_W \phi_S^*}{\phi_S \phi_S^*} \tag{17.16}$$

$$= \frac{\psi_V \psi_W}{\phi_S}, \tag{17.17}$$

and thus the joint distribution as defined in Eq. **??** is unchanged. Whether or not we have achieved anything useful with the update is as yet unclear; but at least the joint probability has not been altered.

We now pass information from W back to V, using the same update rule. In particular:

$$\phi_S^{**} = \sum_{W \setminus S} \psi_W^* \tag{17.18}$$

$$\psi_V^{**} = \frac{\phi_S^{**}}{\phi_S^*} \psi_V^*. \tag{17.19}$$

(Noting that ψ_W^* is unchanged during this update, we define $\psi_W^{**} = \psi_W^*$).

Note that once again the joint probability $p(x_H, \bar{x}_E)$ remains unaltered by the update.

There is another important property that characterizes the pair of updates. In particular, the potentials ψ_V^{**} and ψ_W^{**} are consistent with respect to their intersection S; that is, they have the same marginals. This is easily verified:

$$\sum_{V \setminus S} \psi_V^{**} = \sum_{V \setminus S} \frac{\phi_S^{**}}{\phi_S^*} \psi_V^* \tag{17.20}$$

$$= \frac{\phi_S^{**}}{\phi_S^*} \sum_{V \setminus S} \psi_V^* \tag{17.21}$$

$$= \frac{\phi_S^{**}}{\phi_S^*} \phi_S^* \tag{17.22}$$

$$= \phi_S^{**} \tag{17.23}$$

$$= \sum_{W \setminus S} \psi_W^{**}. \tag{17.24}$$

Inspecting this derivation, we see that the key steps for achieving consistency are Eqs. 17.14 and 17.19. In the forward pass, from V to W, the algorithm stores the marginal of the V potential in the separator potential. In the backward pass, from W to V, the algorithm divides the V potential by its stored marginal and multiplies the result by the new marginal ϕ_S^{**}. This latter marginal is the marginal of the W potential. The rescaling equation essentially substitutes one marginal for another, thus making the two clique potentials consistent. This is achieved in the context of a symmetric algorithm that passes information in both directions, and leaves the joint probability distribution invariant.

Consider for example the Markov chain in Figure 17.8. Initially, the clique potential on $\{X, Y\}$ is $p(x, y)$, and the clique potential on $\{Y, Z\}$ is $p(z \mid y)$. The first pair of update equations results in the following update:

$$\phi_Y^* = \sum_x p(x, y) = p(y) \tag{17.25}$$

$$\psi_{YZ}^* = \frac{p(y)}{1} p(z \mid y) = p(y, z), \tag{17.26}$$

and we see that the clique potentials have become marginal probabilities. The backward phase in this case is vacuous; marginalizing over $p(y, z)$ yields $p(y)$ again for the separator marginal and the update factor is unity.

Now consider the chain in the case in which evidence is observed. Suppose for simplicity that all nodes are binary, and the evidence is $X = 1$. Incorporating the evidence means taking the slice of the potential on $\{X, Y\}$ in which $X = 1$; i.e., taking the second row of the potential table. The marginalization operation is now a vacuous operation, and we have:

$$\phi_Y^* = p(X = 1, y). \tag{17.27}$$

Performing the update of the $\{Y, Z\}$ potential yields:

$$\psi_{YZ}^* = p(X = 1, y)p(z \mid y) = p(X = 1, y, z). \tag{17.28}$$

Thus our potentials are as follows:

$$\psi_{XY}^* = p(X = 1, y) \tag{17.29}$$

$$\phi_Y^* = p(X = 1, y) \tag{17.30}$$

$$\psi_{YZ}^* = p(X = 1, y, z), \tag{17.31}$$

and we see that we have obtained marginals as before, but these are unnormalized marginals. Normalizing (a local operation), we can readily read off the conditionals $p(y \mid X = 1)$, $p(y \mid X = 1)$, and $p(y, z \mid X = 1)$. Note that once again the backward pass is vacuous.

The reader may wish to try the cases in which evidence $Z = 1$ is available and when both $X = 1$ and $Z = 1$ are observed.

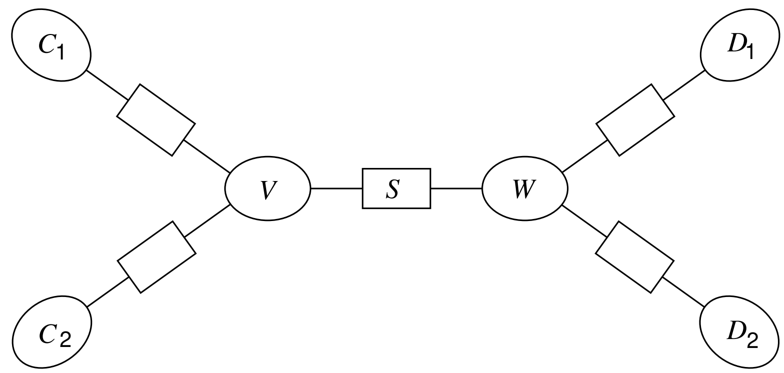

Figure 17.10: A clique tree with explicit representation of the separators. The separators are the intersection of the pair of cliques at the ends of the edge. Thus, for example, $S = V \cap W$.

17.6 Propagation in a clique tree

We now turn to the issue of how to perform local updates when we have multiple overlapping cliques.

In Figure 17.10 we show a clique tree. Each edge in this tree is associated with a separator. Cliques that are neighbors in this tree are subject to the updating procedure described in the previous section.

There are two issues that we must address—how to construct an appropriate clique tree and how to perform the updates so that local consistency obtained between a clique and its neighbor is not ruined by subsequent updates between the clique and other neighbors. In this section we focus on the second issue, returning to the problem of constructing the tree in Section 17.10.

How do we maintain local consistency in a clique tree? Consider again the clique tree shown in Figure 17.10. Suppose that we were to achieve local consistency between V and W using the pair of updates discussed in the previous section, and subsequently we update W based on its other neighbors. The latter updates would generally ruin the consistency that has been achieved between V and W. To ensure that this does not happen, we develop a protocol that constrains the order in which updates are performed.[3]

Let us refer to the update of one clique based on another as a "message-passing" operation. That is, we "pass a message" from V to W by evaluating Eqs. 17.14 and 17.15. In general, as we saw in the previous section, we require a message in both directions in order to render a pair of cliques consistent with each other.

Our problem is to decide when a given clique is allowed to pass a message to one of its neighbors. This problem is solved by the following protocol:

Message-Passing Protocol. *A clique can send a message to a neighboring clique only when it has received messages from all of its other neighbors.*

[3]In fact the protocol is not needed if we are willing to perform redundant steps. If each node is updated repeatedly (for example in parallel), consistency-ruining steps will eventually be corrected (see Exercise ??).

For example, in Figure 17.10, we can send a message from W to V only when W has received messages from its other neighbors D_1 and D_2.

An easy argument establishes the correctness of the protocol. Consider the moment in time at which W has received all of the messages from its other neighbors, and is sending a message to V. There are two cases to consider: either V has not yet sent its message to W, or V has already sent its message to W. In the latter case, we know that V has already received messages from all of its other neighbors. The message from W to V renders the cliques consistent. Neither clique receives any additional messages, thus consistency is maintained. In the former case, W sends a message to V, storing its marginal on S, and waits. At some later time, V will have received all of the messages from its other neighbors and will send a message to W. This message will utilize the stored marginal and render W consistent with V. Neither clique will undergo any additional updates and consistency is maintained.

Although our protocol is correct, is it realizable? Are there message-passing algorithms that realize the protocol and ensure that a message is passed in both directions between every pair of cliques?

There are in fact many message-passing algorithms that realize the protocol; their existence is a simple consequence of the recursive definition of a tree. One way to obtain such algorithms is based on designating one clique in the tree as the root. Once a root of the clique tree is designated, the tree becomes an oriented tree with each leaf having a unique path to the root. Clearly each leaf can send a message inward at any time. Interior nodes send a message toward the root once they have received messages from all of their children. Once all messages have arrived at the root, we propagate messages outward to the leaves.

More formally, we define the following pair of recursive procedures:

```
CollectEvidence( node )
    begin
        for each child of node
            begin
                Update( node, CollectEvidence( child ) )
            end
        return( node )
    end

DistributeEvidence( node )
    begin
        for each child of node
            begin
                Update( child, node )
            DistributeEvidence( child )
        end
    end
```

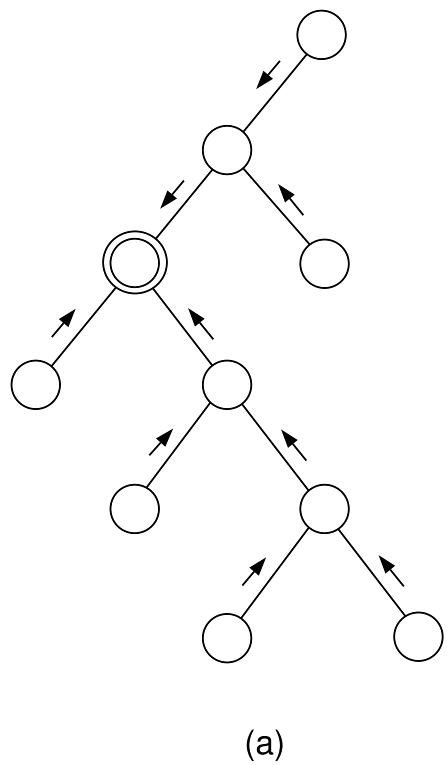

Figure 17.11: (a) The message-passing resulting from a call of COLLECTEVIDENCE at the root node (the doubly-circled node). (b) The message-passing resulting from a call of DISTRIBUTEEVIDENCE at the root node.

where UPDATE(V, W) is a routine that invokes the pair of equations Eq. 17.14 and 17.15. Calling COLLECTEVIDENCE(root) followed by DISTRIBUTEEVIDENCE(root) causes messages to propagate inward to the root and outward to the leaves.

Theorem 1 *The* COLLECTEVIDENCE *and* DISTRIBUTEEVIDENCE *recursions respect the Message-Passing Protocol.*

Proof. When COLLECTEVIDENCE is called at a node, the node calls all of its other neighbors and waits on return messages from those nodes before returning a message back to its caller. Thus COLLECTEVIDENCE obeys the protocol.

After COLLECTEVIDENCE has run, each node has received a message from all of its neighbors except its parent. Once it receives a message from its parent it is free to send messages to any other node. DISTRIBUTEEVIDENCE sends a message from a parent to its child before calling itself on that child. Thus DISTRIBUTEEVIDENCE respects the protocol. $\square$

Consider the example shown in Figure 17.11, where the doubly-circled node is designated as

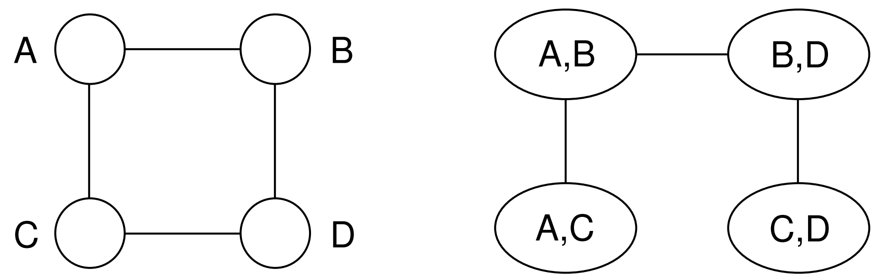

Figure 17.12: An undirected graphical model and a corresponding clique tree.

the root node. A call of CollectEvidence results in messages proceeding inward as shown in Figure 17.11, and a call of DistributeEvidence results in the outward-going messages shown in Figure 17.11. Note that it is clear that one and only one message is passed in both directions between every pair of cliques.

17.7 The junction tree property

At this point we have developed most of the machinery associated with the junction tree algorithm, and we are in the position to describe recursive inference algorithms for some non-trivial graphical models. In fact the machinery discussed thus far is sufficient to handle all of the models that we considered in Part I. In particular, an impatient reader could jump to Chapter 18 to see how the algorithm specializes to the case of the HMM and the state-space model. Both of those cases involve a rather obvious choice for the tree of cliques, and given a particular choice of root node, the recursive algorithms that we developed in earlier chapters fall out naturally from CollectEvidence and DistributeEvidence.

Despite this heady success, we have as yet no theoretical guarantee that the algorithm is correct for general graphical models. In fact it turns out that the algorithm as developed thus far is *not* correct for general graphical models. In this section we identify the (last) problem that must be addressed. We should emphasize at the outset that the problem is essentially a data structure problem involving the construction of the clique tree. There is in fact no problem with our marginalizing and rescaling equations, nor with our Message-Passing Protocol. It suffices to get the data structure right.

To see that our labor is not yet finished, consider the undirected graphical model shown in Figure 17.12. There are four cliques in this graph. A particular choice of clique tree is shown in Figure 17.12. Note that this clique tree has a problematic feature. In particular, the node C appears in two different cliques in the tree and these cliques are not neighbors. Given that our algorithm only enforces local consistency, there is no guarantee that the two cliques containing C will be consistent. Indeed, if the leftmost clique that contains C is changed (e.g., by the introduction of evidence), there is no mechanism to insure that this information will flow to the rightmost clique that contains C. In general, local consistency does not imply global consistency.

Note that the lack of global consistency does not imply that we have an incorrect representation of the joint probability distribution. Indeed, as we saw earlier, the junction tree algorithm does not alter the joint probability, and thus we maintain a correct representation of the joint throughout. What we fail to achieve in Figure 17.12 is locality—the clique potentials correctly represent the joint probability, but they are not local marginal probabilities.

The reader can verify that there is no alternative clique tree that avoids the problem. All clique trees have a pair of nodes that lie in non-neighboring cliques.

A clue to understanding the problem comes from observing that the elimination algorithm would unavoidably create new links in the graph in Figure 17.12; e.g., eliminating C would connect A and B. Another way to put the problem is that there is no way to choose an elimination ordering such that the elimination cliques are contained within the cliques of the original graph.

While this argument based on elimination provides insight, we prefer to restate the problem directly in terms of properties of clique trees. To do so, we articulate a property that rules out the problematic configurations of the kind that we saw in Figure 17.12. The relevant property is known as the *junction tree property*:

The junction tree property. A clique tree possesses the *junction tree property* if for every pair of cliques V and W, all cliques on the (unique) path between V and W contain $V \cap W$.

A clique tree that possesses the junction tree property is referred to as a *junction tree.*

The consequences of the junction tree property for inference are as follows. If a node A appears in two cliques in a junction tree, then A is contained in every clique along the path between these two cliques. If the cliques along the path are *pairwise* consistent with respect to A then they will be *jointly* consistent with respect to A. *In a junction tree, local consistency implies global consistency.*

This argument implies that if we are fortunate enough to have a clique tree that is a junction tree, and if we run the message-passing procedure as described in the previous section, we achieve not only local consistency but also global consistency. We can get the same answer for any node A by consulting any potential that contains A.

Recall however that our goal is to obtain a set of potentials that are not only consistent, but are also marginals—that is, each potential represents the marginal probability of the nodes in its clique. It is conceivable that the junction tree could be consistent, but the potentials would not be marginals. In fact, somewhat surprisingly, this cannot be the case. In a junction tree, the junction tree algorithm not only achieves global consistency, but it yields the sought-after clique marginals as well. To prove this important result we require the following lemma.

Lemma 1 *Let C be a leaf in a junction tree for a graph with vertex set V. Let S be the associated separator (see Figure 17.13). Let $R = C \backslash S$ be the set of nodes in C but not in the separator, and let $U = V \backslash C$ be the set of nodes in V but not in C. Then:*

$$R \perp\!\!\!\perp U \mid S \tag{17.32}$$

Proof. Suppose, by way of contradiction, that $A \in R$ has a neighbor $N \in U$. Consider the maximal complete subset containing both A and N. This clique is not C because $N \notin C$. However, A cannot be contained in any clique other than C because A would have to belong to S as well, by

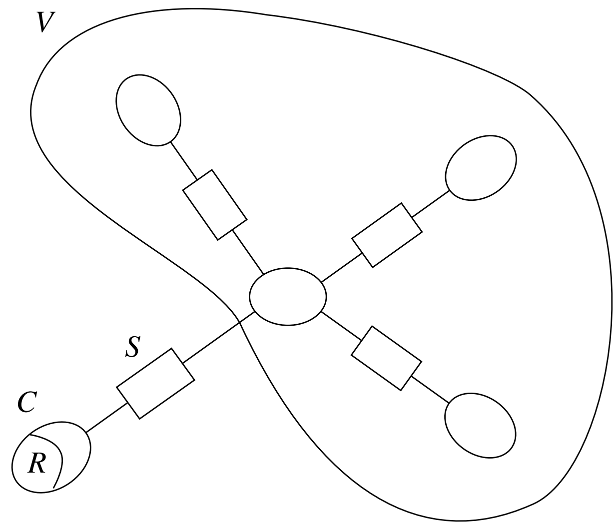

Figure 17.13: The "residual" set $R = C\backslash S$ is the set of nodes in C that are not in S, and, by the junction tree property, also not in U.

the junction tree property, and nodes in R are not in S by definition. Thus no such N exists and S must therefore separate A from U. Since A is arbitrary, S separates R from T. □

We now state and prove our main result.

Theorem 2 *Let the probability $p(x_H, \bar{x}_E)$ be represented by the clique potentials ψ_C and separator potentials ϕ_S of a junction tree. When the junction tree algorithm terminates, the clique potentials and separator potentials are proportional to local marginal probabilities. In particular:*

$$\psi_C = p(x_C, \bar{x}_E) \tag{17.33}$$

$$\phi_S = p(x_S, \bar{x}_E) \tag{17.34}$$

Proof. The separators are subsets of the cliques. That the separator potentials are proportional to marginals therefore follows from the fact that they are consistent with the clique potentials. Thus we need only prove the result for the clique potentials.

The proof is a proof by induction. The result holds for the base case of a single clique by definition. Let us suppose that the result holds for junction trees of N or fewer cliques, and consider a junction tree with $N+1$ cliques.

We choose a clique $\tilde{C}$ that is a leaf in the junction tree. Let $\tilde{S}$ be the corresponding separator, let $\tilde{R} = \tilde{C}\backslash\tilde{S}$ and let $\tilde{T} = V\backslash\tilde{C}$. We also define analogous quantities in which the evidence variables are omitted. In particular, let $C = \tilde{C}\backslash E$, $R = \tilde{R}\backslash E$ and $T = \tilde{T}\backslash E$. By Lemma 1 we have:

$$p(x_H, \bar{x}_E) = p(x_R, x_S, x_T, \bar{x}_E) = p(x_R \,|\, x_S, \bar{x}_E)p(x_S, x_T, \bar{x}_E). \tag{17.35}$$

Summing both sides over R, we obtain:

$$
\begin{aligned}
p(x_S, x_T, \bar{x}_E) &= \sum_R p(H, \bar{x}_E) && (17.36)\\
&= \sum_R \frac{\prod_C \psi_C(x_C)}{\prod_S \phi_S(x_S)} && (17.37)\\
&= \sum_R \frac{\psi_C}{\phi_S} \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} && (17.38)\\
&= \frac{\sum_R \psi_C}{\phi_S} \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} && (17.39)\\
&= \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} && (17.40)
\end{aligned}
$$

where Eq. 17.40 follows from the fact that C and S are consistent and thus $\sum_R \psi_C = \phi_S$.

Eq. 17.40 shows that $p(x_S, x_T, \bar{x}_E)$ is represented by the clique potentials and separator potentials on the junction tree over $S \cup T$. By the induction hypothesis, after a full round of message passing the clique potentials on this junction tree are equal to marginals.

It remains to show that the clique potential on C is a marginal. Let D be the neighbor of C in the junction tree. By consistency we have $\phi_S(x_S) = \sum_{D \setminus S} \psi_D(x_D)$. We have $\psi_D = p(x_D, \bar{x}_E)$ and thus $\psi_S(x_S) = p(x_S, \bar{x}_E)$. Thus:

$$
\begin{aligned}
p(x_R \mid x_S, \bar{x}_E) &= \frac{\psi_C(x_C)}{\phi_S(x_S)} && (17.41)\\
&= \frac{\psi_C(x_C)}{p(x_S, \bar{x}_E)} && (17.42)
\end{aligned}
$$

which implies $\psi_C(x_C) = p(x_C, \bar{x}_E)$. □

17.8 Triangulated graph ⇒ Junction tree

The junction tree property provides a sufficient condition for the correctness of the junction tree algorithm. What class of graphs have a junction tree? How do we handle graphs that do not have a junction tree?

In this section we present a sufficient condition for a graph to have a junction tree—the condition is that the graph must be *triangulated*. It turns out that triangulation is also a necessary condition for a graph to have a junction tree. In the current section, however, we restrict ourselves to the proof of sufficiency, proving necessity in Appendix A. The Appendix also demonstrates that triangulation is equivalent to *decomposability*; a characterization of graphs that we discussed in Section **??**.

We begin by defining a triangulated graph and then proceed to the proof of sufficiency. The reader willing to accept the proof on faith can read the definition of triangulation in the next paragraph and then skip to the following section without loss of continuity.

Consider a cycle in an undirected graph. A cycle is *chordless* if there are no edges between nodes that are not successors in the cycle. For example, the cycle $A-B-D-C-A$ in Figure 17.12 is chordless because there is no edge between A and C or between B and D. A graph is said to be *triangulated* if there are no chordless cycles in the graph.

Our first stop in the proof of sufficiency is a simple lemma that shows that triangulated graphs can be decomposed into three subsets with special properties.

Lemma 2 *Let $\mathcal{G} = (V, E)$ be a noncomplete triangulated graph with at least three nodes. Then there exists a decomposition of V into disjoint sets A, B and S such that S separates A and B and S is complete.*

Proof. Choose a pair of nonadjacent nodes α and β. Let S be the minimal set of nodes such that any path from α to β passes through S. Let A be the set of nodes reachable from α when S is removed and similarly let B be the set of nodes reachable from β when S is removed. Clearly these two sets are separated by S. We need only establish that S is complete.

Let C and D be nodes in S. Since S is minimal, there is a path from α to C and from α to D; thus there is a path from C to D in $A \cup S$. Take the shortest such path. Similarly take the shortest path joining C to D in $B \cup S$. Link these paths to obtain a cycle. This cycle must have a chord. This chord must be an edge between C and D, by our choice of shortest paths. Thus C and D are neighbors. □

We also require the notion of a simplicial node. A node is *simplicial* if all of its neighbors are connected. The following lemma guarantees the existence of simplicial nodes in triangulated graphs.

Lemma 3 *Every triangulated graph that contains at least two nodes has at least two simplicial nodes. If the graph is not complete, then these nodes can be chosen to be nonadjacent.*

Proof. We again use induction and again the base case is trivial. Consider a triangulated graph $\mathcal{G}$ with $N+1$ nodes. If the graph is complete then all nodes are simplicial. Otherwise we use Lemma 2 to decompose the graph into disjoint sets A, B and S. The subgraphs $A \cup S$ and $B \cup S$ cannot contain any chordless cycles (because any such cycles would also be chordless in $\mathcal{G}$), and thus they are both triangulated. The induction hypothesis implies the existence of two simplicial nodes in $A \cup S$. If $A \cup S$ is not complete these can be taken to be nonadjacent, and, given that S is complete, one of the two nodes can be taken to be in A. Otherwise, pick any node in A. Similarly, the induction hypothesis implies the existence of two simplicial nodes in $B \cup S$, and one of these can be taken in B. Given that A and B are separated by S, the two nodes that we have selected are simplicial in $\mathcal{G}$ and they are also nonadjacent. □

We now demonstrate that triangulation implies the existence of a clique tree with the junction tree property.

Theorem 3 *All triangulated graphs have a junction tree.*

Proof. We once again use induction and once again the base case is trivial. Consider a graph $\mathcal{G}$ with $N+1$ nodes. By Lemma 3, the graph has at least one simplicial node α.

Removing a simplicial node from a triangulated graph yields a triangulated graph, because no chordless cycles can be created. Thus by the induction hypothesis, the graph with α removed has a junction tree T. We construct a junction tree for $\mathcal{G}$ from T.

Let C denote the clique formed by α and its neighbors. If $C \backslash \alpha$ is a clique in T, then simply add α to that clique; T with the augmented clique is a junction tree for $\mathcal{G}$.

If $C \backslash \alpha$ is not a clique D in T, then it is a subset of a clique D in T. Add C as a new leaf node for T, with a link to D and a separator set $S = C \backslash \alpha$. The result is a junction tree. This is established by noting that (1) α is contained only in C and therefore cannot violate the junction tree property; and (2) all other nodes in C are in S and in D and therefore cannot violate the junction tree property. $\square$

17.9 Elimination $\Rightarrow$ Triangulation

In this section we show that UndirectedGraphEliminate can be viewed as a procedure for creating a triangulated graph. This result will show us how to deal with nontriangulated graphs within the junction tree framework. It also allows us to demonstrate that the elimination algorithm is a special case of the junction tree algorithm.

Recall that UndirectedGraphEliminate is a simple iterative algorithm that successively eliminates the nodes in a graph by (1) connecting the (remaining) neighbors of the node and (2) removing the node and its edges from the graph. The input to the algorithm is a graph and an elimination ordering.

Theorem 4 UndirectedGraphEliminate *yields a triangulated graph.*

Proof. We prove the theorem by induction. The base case is a graph with a single node, which is obviously triangulated. Suppose now that the hypothesis holds for graphs with N or fewer nodes and consider a graph with $N+1$ nodes. Eliminating a node results in a graph with N nodes, which cannot contain a chordless cycle by the induction hypothesis. Moreover, it is not possible to form a chordless cycle involving the eliminated node, because the elimination step connects all of the neighbors of the node. $\square$

Thus the edges added by the UndirectedGraphEliminate algorithm are exactly those that turn a nontriangulated graph into a triangulated graph.

This result suggests the following general approach to dealing with nontriangulated graphs. Given an initial undirected graph (possibly obtained by moralizing a directed graph), we first triangulate the graph using UndirectedGraphEliminate. We are not constrained in our choice of elimination ordering and can use any of a variety of heuristics to choose a "good" elimination ordering; e.g., one that introduces as few extra edges as possible (see Appendix A). Given a triangulation, we construct a junction tree from the triangulated graph and run the message-passing procedure. The algorithm calculates marginal probabilities for all of the cliques of the triangulated graph. Marginals for subsets of these cliques (e.g., individual nodes) can be obtained by further marginalization and normalization of individual potentials.

The correctness of this approach follows from an argument similar to that used to justify moralization. Adding edges to a graph can only decrease the set of conditional independencies associated with the graph and thus expand the set of probability distributions associated with the graph. This implies that the set of probability distributions associated with the triangulation of a graph includes the set of probability distributions associated with the original graph. Solving the inference problem for the triangulated graph solves it for the original graph as well.

Our argument also suggests (correctly) that the elimination algorithm is a special case of the junction tree algorithm. As we ask the reader to show in Exercise **??**, applying the junction tree algorithm to the cliques of the triangulated graph resulting from a given elimination ordering we recover exactly the probabilistic calculations of the elimination algorithm.

It is possible to prove a converse to Theorem 4 showing that for any triangulated graph there exists an ordering such that elimination using that ordering introduces no new edges.[4] Thus, elimination and triangulation are essentially equivalent notions. This does not imply, however, that practical algorithms for triangulation are necessarily best viewed as elimination algorithms. Rather, treating triangulation as a combinatorial optimization problem provides a broader perspective on the problem. In Appendix A, we return to these issues and describe practical algorithms for graph triangulation.

If our goal is to obtain the marginal probabilities of all of the non-evidence nodes in the graph, then the naive elimination algorithm would require us to choose different elimination orderings in which the target node is the final node in the ordering. These different elimination orderings would in general produce incommensurate elimination cliques, and make it difficult, if not impossible, to share the intermediate potentials. The junction tree framework, on the other hand, calculates a single triangulation, in effect using a single elimination ordering. While this ordering may not be optimal for calculating any given individual marginal, the choice of a single ordering makes it possible to share intermediate potentials, and thus supports the efficient calculation of marginals for all cliques in the graph.

17.10 Constructing the junction tree

The results of Section 17.8 show that every triangulated graph has a junction tree. This proof—an existence proof—leaves us just short of our goal. How do we construct a junction tree from a triangulated graph?

It is certainly not the case that every clique tree obtained from a triangulated graph is a junction tree. Consider the triangulated graph shown in Figure 17.14(a). The clique tree in Figure 17.14(b) is not a junction tree (consider node B). A junction tree for this graph is shown in Figure 17.14(c).

The separators in Figure 17.14(b) are $\{C, D\}$ and $\{D\}$, whereas in Figure 17.14(c) the separators are $\{B, D\}$ and $\{C, D\}$. The total cardinality of the separator sets is larger in the latter figure. Intuitively this fact would seem to have something to do with the fact that Figure 17.14(c) possesses the junction tree property while Figure 17.14(b) does not.

To each clique tree T associated with a triangulated graph we can assign a *weight* $w(T)$ given by the sum of the cardinalities of the separator sets in the tree. We show in this section that a clique tree is a junction tree if and only if it has maximal weight, ranging over all possible trees of cliques. There may be several such trees.

[4] See, e.g., Jensen, (1996).

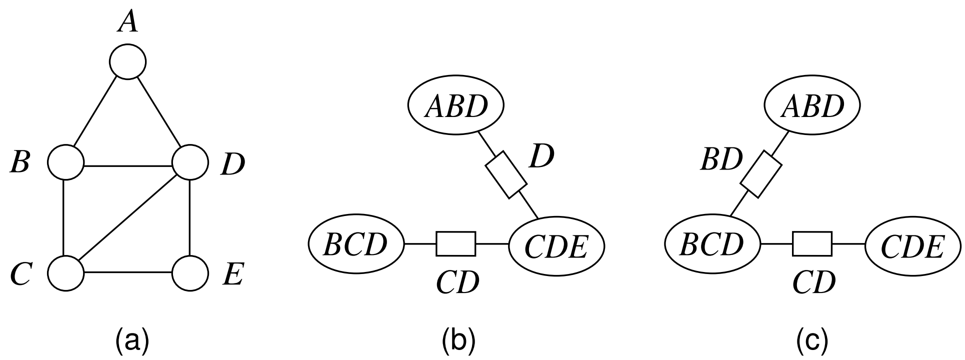

Figure 17.14: (a) A triangulated graph. (b) A clique tree based on (a) that does not have the junction tree property. (c) A clique tree based on (a) that does have the junction tree property.

Our problem is an instance of the classical "maximal spanning tree problem." The problem is readily solved via one of a number of simple greedy algorithms. One solution is given by Kruskal's algorithm: Begin with no edges between the cliques. At each step add an edge that has maximal separator cardinality, ensuring that the resulting graph has no cycles. Once the graph is fully connected (there is a path between any pair of cliques), we have a maximal spanning tree.[5]

Consider a node X_k and a clique tree T with cliques C_i and separators S_j. Consider further the count of the number of times that X_k appears as an element in one the cliques C_i, as well as the count of the number of times that X_k appears as an element in one of the S_j. Clearly these counts are related, and in particular the fact that T is a tree implies that the latter count is no more than the former count less one:

$$\sum_{j=1}^{M-1} 1(X_k \in S_j) \leq \sum_{i=1}^{M} 1(X_k \in C_i) - 1, \tag{17.43}$$

where $1(\cdot)$ is the indicator function and where M is the number of cliques. Moreover, this inequality becomes an equality when the subgraph of T induced by X_k is a tree.

As we have noted earlier, the statement that the subgraph of T induced by a node X_k is a tree is nothing more than a restatement of the junction tree property. Thus we have in Eq. 17.43 an inequality which is indicative of the junction tree property, at least with respect to a single node X_k.

We are now ready to state the theorem linking junction trees and maximal spanning trees.

Theorem 5 *A clique tree T is a junction tree if and only if it is a maximal spanning tree.*

Proof. The total weight of a clique tree is equal to the sum of the cardinalities of its separators.

[5]See Cormen, Leisherson, and Rivest (1990) for a proof of this result. Another approach is given by Prim's algorithm, which maintains a partial tree at each step and iteratively adds nodes to this tree.

Thus we have:

$$
\begin{aligned}
w(T) &= \sum_{j=1}^{M-1} |S_j| && (17.44)\\
&= \sum_{j=1}^{M-1} \sum_{k=1}^{N} 1(X_k \in S_j) && (17.45)\\
&= \sum_{k=1}^{N} \sum_{j=1}^{M-1} 1(X_k \in S_j) && (17.46)\\
&\leq \sum_{k=1}^{N} \left[\sum_{i=1}^{M} 1(X_k \in C_i) - 1 \right] && (17.47)\\
&= \sum_{i=1}^{M} \sum_{k=1}^{N} 1(X_k \in C_i) - M && (17.48)\\
&= \sum_{i=1}^{M} |C_i| - M. && (17.49)
\end{aligned}
$$

Noting that the right-hand side is independent of T, and that the inequality in Eq. 17.47 is an equality if and only if T is a junction tree, we obtain the result. □

17.11 The Hugin algorithm

The algorithm that we have developed in previous sections is known as the "Hugin algorithm," an instance of the general junction tree framework. We summarize the algorithm here. There are five principal steps to the algorithm, the first of which applies only to directed graphs.

- **Moralization**. The moralization step converts a directed graph into an undirected graph. Nodes that have a common child are linked, and directed edges are converted to undirected edges. The local conditional probability of each node is multiplied onto the potential of a clique that contains the node and its parents.

- **Introduction of evidence**. Evidence is introduced by taking slices of the potentials.

- **Triangulation**. The graph is triangulated, using one of several possible algorithms. The potential of each clique of the original graph is multiplied onto the potential of a clique that contains the clique.

- **Construction of junction tree**. A junction tree is constructed by forming a maximal spanning tree from the cliques of the triangulated graph. Separators are introduced and their potentials are initialized to unity.

- **Propagation of probabilities**. Computation proceeds in the junction tree via the following update equations:

$$\phi_S^* = \sum_{V \setminus S} \psi_V \tag{17.50}$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W. \tag{17.51}$$

The updates must respect the Message-Passing Protocol. This can be achieved by designating a root node and calling CollectEvidence and DistributeEvidence from the root. Once the algorithm terminates, the clique potentials and separator potentials are proportional to marginal probabilities. Further marginalization can be performed to obtain the probabilities of singleton nodes or other subsets.

17.12 The Shafer-Shenoy algorithm

There are a number of variations on the junction tree theme. All of these variations have at their core the notion of a triangulated graph and the junction tree property, but the way that propagation proceeds on the junction tree can be different. Some of these variations can provide additional insights into exact inference and provide different pathways for generalizations to approximate inference. Moreover, different variations on junction tree propagation can have different numerical properties or time/space properties. In this section we discuss one such variation—the Shafer-Shenoy algorithm.

The Shafer-Shenoy algorithm can be viewed as a variation on the junction tree framework in which no use is made of separator potentials. While the separator potentials have been useful in providing a simple mechanism for achieving consistency between neighboring cliques, and while we will encounter architectural examples in which separator potentials are particularly useful (cf. Section18.2.4), there is a sense in which separator potentials are redundant (they are simply marginals of the clique potentials) and perhaps they can be disposed with.

Rather than focusing on separator potentials, let us instead focus on the ratios of separator potentials; the quantities that we referred to as "update factors" in our earlier presentation. Recall that in the second step of the message-passing calculation (Eq. 17.15), the clique potential is multiplied by the update factor. What we will show is that a propagation procedure can be based solely on the update factors.

Consider the pair of cliques C_i and C_j in Figure 17.15, with separator $S_{ij} = C_i \cap C_j$. We wish to exchange messages between these cliques so as to implement a junction tree algorithm, and we wish to do so without making use of a potential on the separator S_{ij}. To do so, define $\mu_{ij}(S_{ij})$ as the message sent from C_i to C_j.[6] The Shafer-Shenoy algorithm tells us how to calculate $\mu_{ij}(S_{ij})$ based on the messages arriving at clique C_i from all cliques other than clique C_j:

$$\mu_{ij}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq i} \mu_{ki}(S_{ki}) \tag{17.52}$$

[6]Note that we are using the term "message" in a slightly more specific manner than before; for the Shafer-Shenoy algorithm, we equate "message" with the values $\mu_{ij}(S_{ij})$.

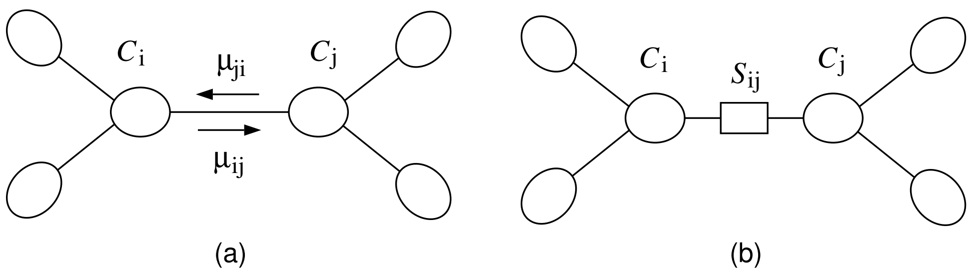

Figure 17.15: (a) A junction tree showing the messages μ_{ij} and μ_{ji} that are passed between cliques C_i and C_j. Note that both messages are functions of the separator S_{ij}. (b) A junction tree showing the separator explicitly.

Once clique C_i has received messages from all of its neighbors, we compute the marginal probability for C_i as follows:

$$p(C_i) \propto \psi_{C_i} \prod_k \mu_{ki}(S_{ki}). \tag{17.53}$$

Equations 17.52 and 17.53 constitute the Shafer-Shenoy algorithm. We now derive this algorithm from the point of view of our earlier junction tree algorithm, thereby proving the correctness of the implicit assertion in Eq. 17.53—that we do in fact obtain the marginal probabilities via this algorithm.

Consider now the pair of cliques C_i and C_j in Figure 17.15(b) with the explicit separator S_{ij}. The connection between the new algorithm and the earlier algorithm is made as follows. Define $\mu_{ij}(S_{ij})$ to be the update factor associated with the update of the link in the direction from C_i to C_j. That is, if the first update of this link proceeds in the i-to-j direction, let:

$$\mu_{ij}(S_{ij}) \triangleq \frac{\phi^*_{S_{ij}}}{\phi_{S_{ij}}}; \tag{17.54}$$

otherwise, let:

$$\mu_{ij}(S_{ij}) \triangleq \frac{\phi^{**}_{S_{ij}}}{\phi^*_{S_{ij}}}. \tag{17.55}$$

In either case, $\mu_{ij}(S_{ij})$ is the update factor arriving at clique C_j from clique C_i. Now note that the final potential at a given clique is the product of its initial potential and all of the update factors arriving from its neighbors. This immediately shows that Eq. 17.53 has the correct form. We have reduced our problem to that of establishing the correctness of Eq. 17.52.

We consider two cases. Suppose first that the initial update of the link between C_i and C_j occurs in the i-to-j direction. For this update to occur it must be the case that C_i has already received updates from all of its other neighbors (the Message-Passing Protocol). Thus at the moment when the update occurs, the value of the potential on C_i must be the product of its initial potential and the update factors from its neighbors C_k, for $k \neq j$. Let us assume (as an inductive hypothesis) that these update factors are correctly given by $\mu_{ki}(S_{ki})$, and consider the update factor that C_i

sends to C_j. From Eq. 17.14 we have:

$$\phi^*_{S_{ij}} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq i} \mu_{ki}(S_{ki}). \tag{17.56}$$

Comparing this with Eq. 17.52, we see that $\mu_{ij}(S_{ij}) = \phi^*_{S_{ij}}$ and, recalling that the initial value of the separator potential, $\phi_{S_{ij}}$, is unity, we have $\mu_{ij}(S_{ij}) = \phi^*_{S_{ij}} \; \phi_{S_{ij}}$ as required.

Now consider the case in which an earlier update has already occurred in the j-to-i direction. In this case, at the moment of the update from C_i to C_j, the potential on C_i must be the product of its initial potential and the update factors from *all* of its neighbors, including C_j. Thus, from Eq. 17.14 we have:

$$\begin{aligned}
\phi^{**}_{S_{ij}} &= \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k} \mu_{ki}(S_{ki}) && (17.57)\\
&= \sum_{C_i \setminus S_{ij}} \psi_{C_i} \mu_{ji}(S_{ji}) \prod_{k \neq j} \mu_{ki}(S_{ki}) && (17.58)\\
&= \sum_{C_i \setminus S_{ij}} \psi_{C_i} \frac{\phi^*_{S_{ij}}}{\phi_{S_{ij}}} \prod_{k \neq j} \mu_{ki}(S_{ki}) && (17.59)\\
& && (17.60)
\end{aligned}$$

and this yields:

$$\frac{\phi^{**}_{S_{ij}}}{\phi^*_{S_{ij}}} = \sum_{C_i \setminus S_{ij}} \prod_{k \neq j} \mu_{ki}(S_{ki}), \tag{17.61}$$

where we again use the fact that $\phi_{S_{ij}} \equiv 1$. Comparing this result with Eq. 17.52, we see that $\mu_{ij}(S_{ij}) = \phi^{**}_{S_{ij}} / \phi^*_{S_{ij}}$ as required.

17.13 Computational complexity

In this section we discuss the computational complexity of the junction tree algorithm. For concreteness we focus on the Hugin algorithm and consider the computational complexity of the Shafer-Shenoy algorithm in the exercises.

It is important to distinguish between two phases of the junction tree algorithm. The first phase, which we will refer to as the *compilation phase*, involves moralization, triangulation and the maximal spanning tree algorithm. The second phase, the *propagation phase*, involves the introduction of evidence and message-passing on the junction tree.

The compilation phase is an "off-line" phase, occuring once for a given graphical model. The algorithms in the propagation phase are "on-line," running each time a new set of conditional probabilities is desired.

Moralization is clearly a computationally tractable procedure. Letting N denote the number of nodes in the graph, and M the number of edges, moralization runs in time $O(N + M)$.

Moreover, the maximal spanning tree problem is computationally tractable. This is a well-studied problem and the computational complexity results are classical. In particular, the run time of Kruskal's algorithm is $O(N^2)$ and the run time of Prim's algorithm is $O(N^2)$.[7]

Let us turn to the triangulation problem. If we are not concerned with optimality (e.g., finding a junction tree with the smallest maximal clique, or the smallest number of edges), then finding a triangulation is computationally tractable. In particular, the run time of UndirectedGraphEliminate is easily seen to be $O(XXX)$. The problem of finding an optimal junction tree, however, is an NP-hard problem, under any of a number of definitions of optimality. We discuss this intractability result in more detail in Appendix A.

The fact that triangulation is an off-line phase of the junction tree algorithm tempers some of the concern that accompanies the NP-hardness result. Moreover, as we discuss in Appendix A, there are heuristic algorithms available for triangulation that perform reasonably well in empirical experiments. One may be willing to pay the cost of allowing one of these algorithms to run for a substantial time to obtain a good triangulation. Finally, it is important to be aware that for many graphical models the initial graph is sufficiently dense that even the optimal triangulation, if it could be found, would have a large number of edges or a large maximal clique size. It is the size of these cliques, which impacts the second phase of the junction tree algorithm, which is generally the key practical limitation in using the algorithm.

The second phase of the algorithm involves conditioning and message-passing. Conditioning is a straightforward procedure that simply annotates each clique with the indices that are to be held fixed in the slice corresponding to the conditioning variables. We therefore turn to the message-passing procedure.

Each step of the message-passing procedure involves the marginalization and rescaling of clique potentials. Let us suppose that these potentials are represented nonparametrically, as tables. This is a worst-case assumption, and specific parametric representations of the clique potentials may give more favorable complexity results. Marginalizing a table requires us to access each entry in the table, and thus the number of operations scales as the number of entries in the table. The number of such entries is exponential in the number of variables in the corresponding clique. This exponentiality is the key determinant of the computational complexity of the junction tree algorithm.

Rescaling a potential again involves accessing each entry in the affected clique potential, and thus is again exponential in the number of variables in the clique.

The number of cliques in a junction tree is no more than N, the number of nodes in the underlying graph (assuming that we use maximal cliques). Thus the number of separators is bounded above by $N-1$, and we have at most $2N-1$ messages flowing in a run of the Hugin algorithm. Each message involves two operations on clique potentials—a marginalization operation and a rescaling operation. In summary, a complete run of the Hugin algorithm involves at most $4N-2$ such operations. Given that the size of a clique can be as large as the number of nodes N, the exponentiality of an individual marginalization or rescaling operation dominates the computational complexity.

It is of interest to compare the number of operations needed to obtain the marginal probabilities

[7]See, e.g., Cormen Leisherson, and Rivest (1990).

Figure 17.16: XXX

of all of the nodes in the graph—obtained via the junction tree algorithm—to the number of operations needed to obtain the marginal probabilities of a single node in the graph—obtained via the elimination algorithm. The latter algorithm is special case; just run CollectEvidence. Touches each potential once.

17.14 Generalized marginalization

One of the virtues of the junction tree framework is its clear distinction between the graph-theoretic and the algebraic machinery involved in probabilistic inference. The algebraic machinery that we utilized in deriving the algorithm was elementary—our proofs reposed on the associative, commutative and distributive laws of arithmetic. As we discuss in this section, if we replace the specific algebraic operators that we used with other operators that obey these same laws, we find that the junction tree framework extends readily to a wide class of other problems involving factorized algebraic expressions, of which probabilistic inference is a special case.

17.14.1 Maximum probability configurations

In Section **??** we discussed the Viterbi algorithm for hidden Markov models. Given an observation sequence, this algorithm returns a single configuration of the hidden states that has maximal probability. In this section we describe a "generalized Viterbi algorithm" that computes most probable configurations for arbitrary graphical models.

That we need to do essentially no additional work to derive such an algorithm is suggested by returning to the example in Figure 17.16. Let us find a set of values of the nodes—a *configuration*—that maximizes the joint probability $p(x_1, x_2, \ldots, x_6)$.

The first few steps of the calculation are as follows:

$$\begin{aligned}\max_x p(x) &= \max_{x_1}\max_{x_2}\max_{x_3}\max_{x_4}\max_{x_5}\max_{x_6} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2)p(x_5 \mid x_3)p(x_6 \mid x_2, x_5) \\ &= \max_{x_1} p(x_1)\max_{x_2} p(x_2 \mid x_1)\max_{x_3} p(x_3 \mid x_1)\max_{x_4} p(x_4 \mid x_2)\max_{x_5} p(x_5 \mid x_3)\max_{x_6} p(x_6 \mid x_2, x_5).\end{aligned}$$

Computing the maximum of $p(x_6 \mid x_2, x_5)$ with respect to x_6 yields an intermediate factor that is a function of x_2 and x_5. This factor is then retained until needed in a subsequent maximization, in this case the maximization over x_5.

The sequence of steps continue in an identical manner to those that we carried out in our development of the elimination algorithm in Chapter 3. Clearly, from a symbolic point of view, the computation is the same. In particular, the graphical consequences of the maximization operator are identical to those of our earlier calculations with the summation operator.

In the light of this example, let us consider replacing "sum" with "max" in the junction tree algorithm. The only step in the algorithm that specifically refers to summation is the marginalization step in Eq. (17.14). Changing this step to maximization, we have:

$$\phi_S^* = \max_{V \setminus S} \psi_V \tag{17.62}$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W, \tag{17.63}$$

where the rescaling step is unchanged.

In essence we now obtain an inference algorithm based on a generalized notion of marginalization. All of the steps that we took in deriving the junction tree algorithm go through as before, given that the maximization operator has the same commutativity and associativity properties as summation, and given that maximization distributes over multiplication just as summation distributes over multiplication.

What do we obtain from this algorithm? Recall that our key result is that contained in Theorem 2, where we showed that at the end of the junction tree procedure, each clique potential is equal to its marginal probability. Here "marginal" means that the random variables not contained in the clique have been "summed out." If we replace summation by maximization, we obtain the same result, but now "marginal" means that the random variables not contained in the clique have been "maximized out." Thus, we must have

$$\psi_C(x_C) = \max_{V \setminus C} p(x). \tag{17.64}$$

We interpret the resulting entries in the clique potential as containing the values of the maximal probability attainable for each possible configuration of the random variables X_C. Maximizing over these values, we obtain the actual configuration

We could also take one or more of the variables to be evidence variables and maximize the conditional probability distribution of the remaining variables; this would simply involve holding the evidence variables fixed.

17.14.2 Appendix A. Decomposable $\equiv$ Triangulated $\equiv$ Junction tree

In Section 17.8 we showed that all triangulated graphs possess a junction tree. For the purpose of devising an inference algorithm, this result suffices, focusing our attention on the problem of finding a triangulation of a graph. It is of interest to know, however, that in a certain sense triangulation is not merely a means to an end, but rather triangulation is forced on us if we wish to avail ourselves of the junction tree property. In particular, in this Appendix we strengthen our earlier result and show that a graph has a junction tree if and only if the graph is triangulated.

We also show that these two properties are equivalent to a third property—*decomposability.* Recall from Section **??** that a graph is *decomposable* if it can be recursively subdivided into sets A, B and S, where S separates A and B, and where S is complete. The equivalence of decomposability and the junction tree property provides an appealing interpretation of the junction tree algorithm as a divide-and-conquer algorithm.

Theorem 6 *All decomposable graphs are triangulated.*

Proof. We prove the result by induction. The base case of a single node is trivial. We assume that the result holds for N or fewer nodes and consider a graph with $N+1$ nodes.

If the graph is complete then it is obviously triangulated. Otherwise, the definition of decomposability implies a decomposition of the graph into sets A, B, and S such that S is complete and S separates A and B. Also, both $A \cup S$ and $B \cup S$ are decomposable. By the induction hypothesis there are no chordless cycles in either $A \cup S$ or $B \cup S$. The only possible chordless cycles must therefore include one or more nodes in both A and B. But such cycles must pass twice through S, and the completeness of S implies that they have a chord. $\square$

Theorem 6 and Theorem 3 together show that all decomposable graphs have a junction tree. We have proved the correctness of the junction tree algorithm for the class of decomposable graphs.

We now show a stronger result, namely that decomposability, triangulation and the junction tree property are equivalent. This implies that the junction tree algorithm is correct *only* for the class of decomposable graphs.

Theorem 7 *The following are equivalent characterizations of an undirected graph $\mathcal{G}$:*

(D) G is decomposable.

(T) G is triangulated.

(J) G has a junction tree.

Proof. We have already shown that (D) implies (T) implies (J). Thus we can prove the theorem by showing that (J) implies (D).

The proof is a proof by induction. In the base case $\mathcal{G}$ has a single clique and is decomposable by definition. Suppose that the theorem holds for junction trees with N or fewer cliques and consider a junction tree T for $\mathcal{G} = (X, E)$ with $N+1$ cliques.

Let C be a leaf node in T with separator S. Define $R = C \backslash S$ (recall Figure 17.13). Consider the disjoint sets R, $X \backslash C$ and S. We show that these sets are a decomposition of $\mathcal{G}$.

Lemma 1 implies that S separates R and $X \backslash C$. That S is complete follows from the fact that S is the intersection of a pair of cliques.

To show that $\mathcal{G}$ is decomposable it remains to show that $C = R \cup S$ and $X \backslash R = (X \backslash C) \cup S$ are decomposable. We show that both subsets have junction trees and conclude by the induction hypothesis that they are decomposable.

That C has a junction tree follows immediately because it is a single clique.

Consider the effect on the junction tree T of the removal of nodes in R. Each node in R is contained only in C and its neighbors are therefore fully connected (i.e., nodes in R are simplicial). Removing any such node therefore leaves the remaining nodes in C fully connected, and thus C remains a clique and the junction tree T is unaltered. When all nodes in R have been removed, all that remains of C is the separator S, which is a subset of the neighboring clique in T. By simply pruning the C clique and its separator S from T we therefore obtain a junction tree for $X \backslash R$. $\square$

17.15 Historical remarks and bibliography