20 *Exact inference for graphical models*

20.1 Introduction

In Section 17.4.3, we discussed the forwards-backwards algorithm, which can exactly compute the posterior marginals $p(x_t|\mathbf{v}, \boldsymbol{\theta})$ in any chain-structured graphical model, where $\mathbf{x}$ are the hidden variables (assumed discrete) and $\mathbf{v}$ are the visible variables. This algorithm can be modified to compute the posterior mode and posterior samples. A similar algorithm for linear-Gaussian chains, known as the Kalman smoother, was discussed in Section 18.3.2. Our goal in this chapter is to generalize these exact inference algorithms to arbitrary graphs. The resulting methods apply to both directed and undirected graphical models. We will describe a variety of algorithms, but we omit their derivations for brevity. See e.g., (Darwiche 2009; Koller and Friedman 2009) for a detailed exposition of exact inference techniques for discrete directed graphical models.

20.2 Belief propagation for trees

In this section, we generalize the forwards-backwards algorithm from chains to trees. The resulting algorithm is known as **belief propagation** (**BP**) (Pearl 1988), or the **sum-product algorithm**.

20.2.1 Serial protocol

We initially assume (for notational simplicity) that the model is a pairwise MRF (or CRF), i.e.,

$$p(\mathbf{x}|\mathbf{v}) = \frac{1}{Z(\mathbf{v})} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(x_s, x_t) \tag{20.1}$$

where ψ_s is the local evidence for node s, and ψ_{st} is the potential for edge $s - t$. We will consider the case of models with higher order cliques (such as directed trees) later on.

One way to implement BP for undirected trees is as follows. Pick an arbitrary node and call it the root, r. Now orient all edges away from r (intuitively, we can imagine "picking up the graph" at node r and letting all the edges "dangle" down). This gives us a well-defined notion of parent and child. Now we send messages up from the leaves to the root (the **collect evidence** phase) and then back down from the root (the **distribute evidence** phase), in a manner analogous to forwards-backwards on chains.

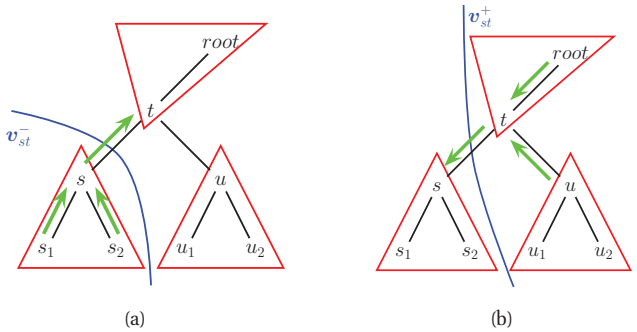

Figure 20.1 Message passing on a tree. (a) Collect-to-root phase. (b) Distribute-from-root phase.

To explain the process in more detail, consider the example in Figure 20.1. Suppose we want to compute the belief state at node t. We will initially condition the belief only on evidence that is at or below t in the graph, i.e., we want to compute $\text{bel}_t^-(x_t) \triangleq p(x_t|\mathbf{v}_t^-)$. We will call this a "bottom-up belief state". Suppose, by induction, that we have computed "messages" from t's two children, summarizing what they think t should know about the evidence in their subtrees, i.e., we have computed $m_{s\to t}^-(x_t) = p(x_t|\mathbf{v}_{st}^-)$, where $\mathbf{v}_{st}^-$ is all the evidence on the downstream side of the $s-t$ edge (see Figure 20.1(a)), and similarly we have computed $m_{u\to t}(x_t)$. Then we can compute the bottom-up belief state at t as follows:

$$\text{bel}_t^-(x_t) \triangleq p(x_t|\mathbf{v}_t^-) = \frac{1}{Z_t}\psi_t(x_t) \prod_{c\in \text{ch}(t)} m_{c\to t}^-(x_t) \tag{20.2}$$

where $\psi_t(x_t) \propto p(x_t|\mathbf{v}_t)$ is the local evidence for node t, and Z_t is the local normalization constant. In words, we multiply all the incoming messages from our children, as well as the incoming message from our local evidence, and then normalize.

We have explained how to compute the bottom-up belief states from the bottom-up messages. How do we compute the messages themselves? Consider computing $m_{s\to t}^-(x_t)$, where s is one of t's children. Assume, by recursion, that we have computed $\text{bel}_s^-(x_s) = p(x_s|\mathbf{v}_s^-)$. Then we can compute the message as follows:

$$m_{s\to t}^-(x_t) = \sum_{x_s} \psi_{st}(x_s, x_t)\text{bel}_s^-(x_s) \tag{20.3}$$

Essentially we convert beliefs about x_s into beliefs about x_t by using the edge potential ψ_{st}.

We continue in this way up the tree until we reach the root. Once at the root, we have "seen" all the evidence in the tree, so we can compute our local belief state at the root using

$$\text{bel}_r(x_r) \triangleq p(x_r|\mathbf{v}) = p(x_t|\mathbf{v}_r^-) \propto \psi_r(x_r) \prod_{c\in \text{ch}(r)} m_{c\to r}^-(x_r) \tag{20.4}$$

This completes the end of the upwards pass, which is analogous to the forwards pass in an HMM. As a "side effect", we can compute the probability of the evidence by collecting the

normalization constants:

$$p(\mathbf{v}) = \prod_t Z_t \tag{20.5}$$

We can now pass messages down from the root. For example, consider node s, with parent t, as shown in Figure 20.1(b). To compute the belief state for s, we need to combine the bottom-up belief for s together with a top-down message from t, which summarizes all the information in the rest of the graph, $m^+_{t\to s}(x_s) \triangleq p(x_t|\mathbf{v}^+_{st})$, where $\mathbf{v}^+_{st}$ is all the evidence on the upstream (root) side of the $s-t$ edge, as shown in Figure 20.1(b). We then have

$$\text{bel}_s(x_s) \triangleq p(x_s|\mathbf{v}) \propto \text{bel}^-_s(x_s) \prod_{t\in \text{pa}(s)} m^+_{t\to s}(x_t) \tag{20.6}$$

How do we compute these downward messages? For example, consider the message from t to s. Suppose t's parent is r, and t's children are s and u, as shown in Figure 20.1(b). We want to include in $m^+_{t\to s}$ all the information that t has received, except for the information that s sent it:

$$m^+_{t\to s}(x_s) \triangleq p(x_s|\mathbf{v}^+_{st}) = \sum_{x_t} \psi_{st}(x_s, x_t) \frac{\text{bel}_t(x_t)}{m^-_{s\to t}(x_t)} \tag{20.7}$$

Rather than dividing out the message sent up to t, we can plug in the equation of bel_t to get

$$m^+_{t\to s}(x_s) = \sum_{x_t} \psi_{st}(x_s, x_t)\psi_t(x_t) \prod_{c\in \text{ch}(t), c\neq s} m^-_{c\to t}(x_t) \prod_{p\in \text{pa}(t)} m^+_{p\to t}(x_t) \tag{20.8}$$

In other words, we multiply together all the messages coming into t from all nodes except for the recipient s, combine together, and then pass through the edge potential ψ_{st}. In the case of a chain, t only has one child s and one parent p, so the above simplifies to

$$m^+_{t\to s}(x_s) = \sum_{x_t} \psi_{st}(x_s, x_t)\psi_t(x_t) m^+_{p\to t}(x_t) \tag{20.9}$$

The version of BP in which we use division is called **belief updating**, and the version in which we multiply all-but-one of the messages is called **sum-product**. The belief updating version is analogous to how we formulated the Kalman smoother in Section 18.3.2: the top-down messages depend on the bottom-up messages. This means they can be interpreted as conditional posterior probabilities. The sum-product version is analogous to how we formulated the backwards algorithm in Section 17.4.3: the top-down messages are completely independent of the bottom-up messages, which means they can only be interpreted as conditional likelihoods. See Section 18.3.2.3 for a more detailed discussion of this subtle difference.

20.2.2 Parallel protocol

So far, we have presented a serial version of the algorithm, in which we send messages up to the root and back. This is the optimal approach for a tree, and is a natural extension of forwards-backwards on chains. However, as a prelude to handling general graphs with loops, we now consider a parallel version of BP. This gives equivalent results to the serial version but is less efficient when implemented on a serial machine.

The basic idea is that all nodes receive messages from their neighbors in parallel, they then updates their belief states, and finally they send new messages back out to their neighbors. This process repeats until convergence. This kind of computing architecture is called a **systolic array**, due to its resemblance to a beating heart.

More precisely, we initialize all messages to the all 1's vector. Then, in parallel, each node absorbs messages from all its neighbors using

$$\text{bel}_s(x_s) \propto \psi_s(x_s) \prod_{t \in \text{nbr}_s} m_{t \to s}(x_s) \tag{20.10}$$

Then, in parallel, each node sends messages to each of its neighbors:

$$m_{s \to t}(x_t) = \sum_{x_s} \left(\psi_s(x_s) \psi_{st}(x_s, x_t) \prod_{u \in \text{nbr}_s \setminus t} m_{u \to s}(x_s) \right) \tag{20.11}$$

The $m_{s \to t}$ message is computed by multiplying together all incoming messages, except the one sent by the recipient, and then passing through the ψ_{st} potential.

At iteration T of the algorithm, $\text{bel}_s(x_s)$ represents the posterior belief of x_s conditioned on the evidence that is T steps away in the graph. After $D(G)$ steps, where $D(G)$ is the **diameter** of the graph (the largest distance between any two pairs of nodes), every node has obtained information from all the other nodes. Its local belief state is then the correct posterior marginal. Since the diameter of a tree is at most $|\mathcal{V}| - 1$, the algorithm converges in a linear number of steps.

We can actually derive the up-down version of the algorithm by imposing the condition that a node can only send a message once it has received messages from all its other neighbors. This means we must start with the leaf nodes, which only have one neighbor. The messages then propagate up to the root and back. We can also update the nodes in a random order. The only requirement is that each node get updated $D(G)$ times. This is just enough time for information to spread throughout the whole tree.

Similar parallel, distributed algorithms for solving linear systems of equations are discussed in (Bertsekas 1997). In particular, the Gauss-Seidel algorithm is analogous to the serial up-down version of BP, and the Jacobi algorithm is analogous to the parallel version of BP.

20.2.3 Gaussian BP *

Now consider the case where $p(\mathbf{x}|\mathbf{v})$ is jointly Gaussian, so it can be represented as a Gaussian pairwise MRF, as in Section 19.4.4. We now present the belief propagation algorithm for this class of models, follow the presentation of (Bickson 2009) (see also (Malioutov et al. 2006)). We will assume the following node and edge potentials:

$$\psi_t(x_t) = \exp(-\frac{1}{2} A_{tt} x_t^2 + b_t x_t) \tag{20.12}$$

$$\psi_{st}(x_s, x_t) = \exp(-\frac{1}{2} x_s A_{st} x_t) \tag{20.13}$$

so the overall model has the form

$$p(\mathbf{x}|\mathbf{v}) \propto \exp(-\frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x}) \tag{20.14}$$

This is the information form of the MVN (see Exercise 9.2), where $\mathbf{A}$ is the precision matrix. Note that by completing the square, the local evidence can be rewritten as a Gaussian:

$$\psi_t(x_t) \propto \mathcal{N}(b_t/A_{tt}, A_{tt}^{-1}) \triangleq \mathcal{N}(m_t, \ell_t^{-1}) \tag{20.15}$$

Below we describe how to use BP to compute the posterior node marginals,

$$p(x_t|\mathbf{v}) = \mathcal{N}(\mu_t, \lambda_t^{-1}) \tag{20.16}$$

If the graph is a tree, the method is exact. If the graph is loopy, the posterior means may still be exact, but the posterior variances are often too small (Weiss and Freeman 1999).

Although the precision matrix $\mathbf{A}$ is often sparse, computing the posterior mean requires inverting it, since $\boldsymbol{\mu} = \mathbf{A}^{-1}\mathbf{b}$. BP provides a way to exploit graph structure to perform this computation in $O(D)$ time instead of $O(D^3)$. This is related to various methods from linear algebra, as discussed in (Bickson 2009).

Since the model is jointly Gaussian, all marginals and all messages will be Gaussian. The key operations we need are to multiply together two Gaussian factors, and to marginalize out a variable from a joint Gaussian factor.

For multiplication, we can use the fact that the product of two Gaussians is Gaussian:

$$\mathcal{N}(x|\mu_1, \lambda_1^{-1}) \times \mathcal{N}(x|\mu_2, \lambda_2^{-1}) = C\mathcal{N}(x|\mu, \lambda^{-1}) \tag{20.17}$$

$$\lambda = \lambda_1 + \lambda_2 \tag{20.18}$$

$$\mu = \lambda^{-1}(\mu_1\lambda_1 + \mu_2\lambda_2) \tag{20.19}$$

where

$$C = \sqrt{\frac{\lambda}{\lambda_1\lambda_2}} \exp\left(\frac{1}{2}(\lambda_1\mu_1^2(\lambda^{-1}\lambda_1 - 1) + \lambda_2\mu_2^2(\lambda^{-1}\lambda_2 - 1) + 2\lambda^{-1}\lambda_1\lambda_2\mu_1\mu_2)\right) \tag{20.20}$$

See Exercise 20.2 for the proof.

For marginalization, we have the following result:

$$\int \exp(-ax^2 + bx)dx = \sqrt{\pi/a}\exp(b^2/4a) \tag{20.21}$$

which follows from the normalization constant of a Gaussian (Exercise 2.11).

We now have all the pieces we need. In particular, let the message $m_{s\to t}(x_t)$ be a Gaussian with mean μ_{st} and precision λ_{st}. From Equation 20.10, the belief at node s is given by the product of incoming messages times the local evidence (Equation 20.15) and hence

$$\text{bel}_s(x_s) = \psi_s(x_s) \prod_{t\in\text{nbr}(s)} m_{ts}(x_s) = \mathcal{N}(x_s|\mu_s, \lambda_s^{-1}) \tag{20.22}$$

$$\lambda_s = \ell_s + \sum_{t\in\text{nbr}(s)} \lambda_{ts} \tag{20.23}$$

$$\mu_s = \lambda_s^{-1}\left(\ell_s m_s + \sum_{t\in\text{nbr}(s)} \lambda_{ts}\mu_{ts}\right) \tag{20.24}$$

To compute the messages themselves, we use Equation 20.11, which is given by

$$m_{s\to t}(x_t) = \int_{x_s} \left(\psi_{st}(x_s, x_t)\psi_s(x_s) \prod_{u \in \text{nbr}_s \setminus t} m_{u\to s}(x_s) \right) dx_s \tag{20.25}$$

$$= \int_{x_s} \psi_{st}(x_s, x_t) f_{s\setminus t}(x_s) dx_s \tag{20.26}$$

where $f_{s\setminus t}(x_s)$ is the product of the local evidence and all incoming messages excluding the message from t:

$$f_{s\setminus t}(x_s) \triangleq \psi_s(x_s) \prod_{u \in \text{nbr}_s \setminus t} m_{u\to s}(x_s) \tag{20.27}$$

$$= \mathcal{N}(x_s | \mu_{s\setminus t}, \lambda_{s\setminus t}^{-1}) \tag{20.28}$$

$$\lambda_{s\setminus t} \triangleq \ell_s + \sum_{u \in \text{nbr}(s)\setminus t} \lambda_{us} \tag{20.29}$$

$$\mu_{s\setminus t} \triangleq \lambda_{s\setminus t}^{-1} \left(\ell_s m_s + \sum_{u \in \text{nbr}(s)\setminus t} \lambda_{us}\mu_{us} \right) \tag{20.30}$$

Returning to Equation 20.26 we have

$$m_{s\to t}(x_t) = \int_{x_s} \underbrace{\exp(-x_s A_{st} x_t)}_{\psi_{st}(x_s,x_t)} \underbrace{\exp(-\lambda_{s\setminus t}/2(x_s - \mu_{s\setminus t})^2)}_{f_{s\setminus t}(x_s)} dx_s \tag{20.31}$$

$$= \int_{x_s} \exp\left((-\lambda_{s\setminus t} x_s^2/2) + (\lambda_{s\setminus t}\mu_{s\setminus t} - A_{st}x_t)x_s \right) dx_s + \text{const} \tag{20.32}$$

$$\propto \exp\left((\lambda_{s\setminus t}\mu_{s\setminus t} - A_{st}x_t)^2/(2\lambda_{s\setminus t}) \right) \tag{20.33}$$

$$\propto \mathcal{N}(\mu_{st}, \lambda_{st}^{-1}) \tag{20.34}$$

$$\lambda_{st} = A_{st}^2/\lambda_{s\setminus t} \tag{20.35}$$

$$\mu_{st} = A_{st}\mu_{s\setminus t}/\lambda_{st} \tag{20.36}$$

One can generalize these equations to the case where each node is a vector, and the messages become small MVNs instead of scalar Gaussians (Alag and Agogino 1996). If we apply the resulting algorithm to a linear dynamical system, we recover the Kalman smoothing algorithm of Section 18.3.2.

To perform message passing in models with non-Gaussian potentials, one can use sampling methods to approximate the relevant integrals. This is called **non-parametric BP** (Sudderth et al. 2003; Isard 2003; Sudderth et al. 2010).

20.2.4 Other BP variants *

In this section, we briefly discuss several variants of the main algorithm.

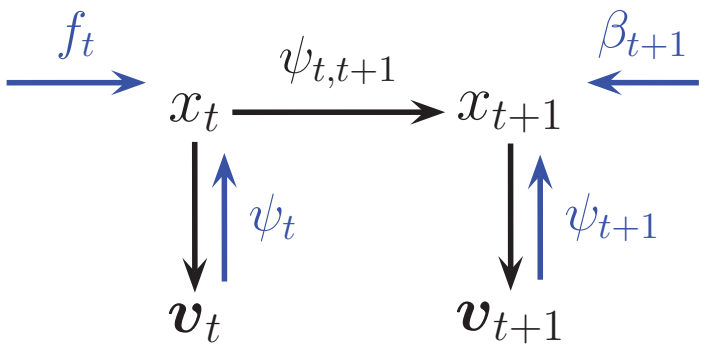

Figure 20.2 Illustration of how to compute the two-slice distribution for an HMM. The ψ_t and ψ_{t+1} terms are the local evidence messages from the visible nodes $\mathbf{v}_t$, $\mathbf{v}_{t+1}$ to the hidde nodes x_t, x_{t+1} respectively; f_t is the forwards message from x_{t-1} and β_{t+1} is the backwards message from x_{t+2}.

20.2.4.1 Max-product algorithm

It is possible to devise a **max-product** version of the BP algorithm, by replacing the $\sum$ operator with the $\max$ operator. We can then compute the local MAP marginal of each node. However, if there are ties, this might not be globally consistent, as discussed in Section 17.4.4. Fortunately, we can generalize the Viterbi algorithm to trees, where we use max and argmax in the collect-to-root phase, and perform traceback in the distribute-from-root phase. See (Dawid 1992) for details.

20.2.4.2 Sampling from a tree

It is possible to draw samples from a tree structured model by generalizing the forwards filtering / backwards sampling algorithm discussed in Section 17.4.5. See (Dawid 1992) for details.

20.2.4.3 Computing posteriors on sets of variables

In Section 17.4.3.2, we explained how to compute the "two-slice" distribution $\xi_{t,t+1}(i,j) = p(x_t = i, x_{t+1} = j|\mathbf{v})$ in an HMM, namely by using

$$\xi_{t,t+1}(i,j) \quad = \quad \alpha_t(i)\psi_{t+1}(j)\beta_{t+1}(j)\psi_{t,t+1}(i,j) \tag{20.37}$$

Since $\alpha_t(i) \propto \psi_t(i)f_t(i)$, where $f_t = p(x_t|\mathbf{v}_{1:t-1})$ is the forwards message, we can think of this as sending messages f_t and ψ_t into x_t, β_{t+1} and ϕ_{t+1} into x_{t+1}, and then combining them with the $\boldsymbol{\Psi}$ matrix, as shown in Figure 20.2. This is like treating x_t and x_{t+1} as a single "mega node", and then multiplying all the incoming messages as well as all the local factors (here, $\psi_{t,t+1}$).

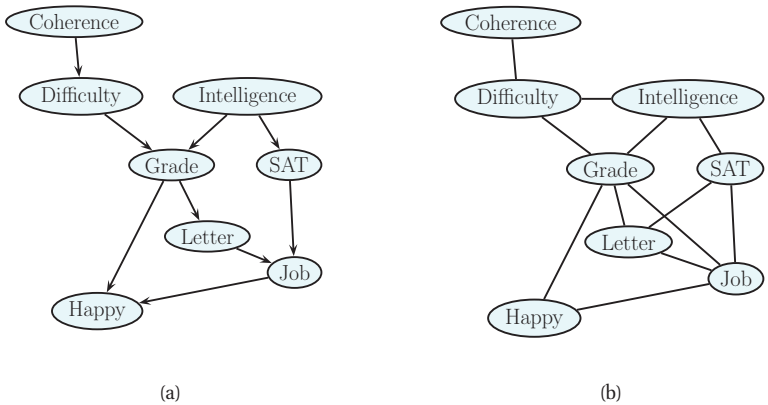

Figure 20.3 Left: The "student" DGM. Right: the equivalent UGM. We add moralization arcs D-I, G-J and L-S. Based on Figure 9.8 of (Koller and Friedman 2009).

20.3 The variable elimination algorithm

We have seen how to use BP to compute exact marginals on chains and trees. In this section, we discuss an algorithm to compute $p(\mathbf{x}_q|\mathbf{x}_v)$ for any kind of graph.

We will explain the algorithm by example. Consider the DGM in Figure 20.3(a). This model, from (Koller and Friedman 2009), is a hypothetical model relating various variables pertaining to a typical student. The corresponding joint has the following form:

$$P(C, D, I, G, S, L, J, H) \tag{20.38}$$

$$= P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, J) \tag{20.39}$$

Note that the forms of the CPDs do not matter, since all our calculations will be symbolic. However, for illustration purposes, we will assume all variables are binary.

Before proceeding, we convert our model to undirected form. This is not required, but it makes for a more unified presentation, since the resulting method can then be applied to both DGMs and UGMs (and, as we will see in Section 20.3.1, to a variety of other problems that have nothing to do with graphical models). Since the computational complexity of inference in DGMs and UGMs is, generally speaking, the same, nothing is lost in this transformation from a computational point of view.[1]

To convert the DGM to a UGM, we simply define a potential or factor for every CPD, yielding

$$p(C, D, I, G, S, L, J, H) \tag{20.40}$$

$$= \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I)\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \tag{20.41}$$

1. There are a few "tricks" one can exploit in the directed case that cannot easily be exploited in the undirected case. One important example is **barren node removal**. To explain this, consider a naive Bayes classifier, as in Figure 10.2. Suppose we want to infer y and we observe x_1 and x_2, but not x_3 and x_4. It is clear that we can safely remove x_3 and x_4, since $\sum_{x_3} p(x_3|y) = 1$, and similarly for x_4. In general, once we have removed hidden leaves, we can apply this process recursively. Since potential functions do not necessary sum to one, we cannot use this trick in the undirected case. See (Koller and Friedman 2009) for a variety of other speedup tricks.

Since all the potentials are **locally normalized**, since they are CPDs, there is no need for a global normalization constant, so $Z = 1$. The corresponding undirected graph is shown in Figure 20.3(b). Note that it has more edges than the DAG. In particular, any "unmarried" nodes that share a child must get "married", by adding an edge between them; this process is known as **moralization**. Only then can the arrows be dropped. In this example, we added D-I, G-J, and L-S moralization arcs. The reason this operation is required is to ensure that the CI properties of the UGM match those of the DGM, as explained in Section 19.2.2. It also ensures there is a clique that can "store" the CPDs of each family.

Now suppose we want to compute $p(J = 1)$, the marginal probability that a person will get a job. Since we have 8 binary variables, we could simply enumerate over all possible assignments to all the variables (except for J), adding up the probability of each joint instantiation:

$$p(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C p(C, D, I, G, S, L, J, H) \tag{20.42}$$

However, this would take $O(2^7)$ time. We can be smarter by **pushing sums inside products**. This is the key idea behind the **variable elimination** algorithm (Zhang and Poole 1996), also called **bucket elimination** (Dechter 1996), or, in the context of genetic pedigree trees, the **peeling algorithm** (Cannings et al. 1978). In our example, we get

$$\begin{aligned} p(J) &= \sum_{L,S,G,H,I,D,C} p(C, D, I, G, S, L, J, H) \\ &= \sum_{L,S,G,H,I,D,C} \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I)\psi_L(L, G) \\ &\quad \times \psi_J(J, L, S)\psi_H(H, G, J) \\ &= \sum_{L,S} \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I)\psi_I(I) \\ &\quad \times \sum_D \psi_G(G, I, D) \sum_C \psi_C(C)\psi_D(D, C) \end{aligned}$$

We now evaluate this expression, working right to left as shown in Table 20.1. First we multiply together all the terms in the scope of the $\sum_C$ operator to create the temporary factor

$$\tau'_1(C, D) = \psi_C(C)\psi_D(D, C) \tag{20.43}$$

Then we marginalize out C to get the new factor

$$\tau_1(D) = \sum_C \tau'_1(C, D) \tag{20.44}$$

Next we multiply together all the terms in the scope of the $\sum_D$ operator and then marginalize out to create

$$\tau'_2(G, I, D) = \psi_G(G, I, D)\tau_1(D) \tag{20.45}$$

$$\tau_2(G, I) = \sum_D \tau'_2(G, I, D) \tag{20.46}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_G(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_G(G,I,D)\tau_1(D)}_{\tau_2(G,I)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \underbrace{\sum_I \psi_S(S,I)\psi_I(I)\tau_2(G,I)}_{\tau_3(G,S)}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \underbrace{\sum_H \psi_H(H,G,J)}_{\tau_4(G,J)} \tau_3(G,S)$$

$$\sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_G \psi_L(L,G)\tau_4(G,J)\tau_3(G,S)}_{\tau_5(J,L,S)}$$

$$\sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_5(J,L,S)}_{\tau_6(J,L)}$$

$$\underbrace{\sum_L \tau_6(J,L)}_{\tau_7(J)}$$

Table 20.1 Eliminating variables from Figure 20.3 in the order C, D, I, H, G, S, L to compute $P(J)$.

Next we multiply together all the terms in the scope of the $\sum_I$ operator and then marginalize out to create

$$\tau'_3(G,I,S) = \psi_S(S,I)\psi_I(I)\tau_2(G,I) \quad (20.47)$$

$$\tau_3(G,S) = \sum_I \tau'_3(G,I,S) \quad (20.48)$$

And so on.

The above technique can be used to compute any marginal of interest, such as $p(J)$ or $p(J,H)$. To compute a conditional, we can take a ratio of two marginals, where the visible variables have been clamped to their known values (and hence don't need to be summed over). For example,

$$p(J=j|I=1,H=0) = \frac{p(J=j,I=1,H=0)}{\sum_{j'} p(J=j',I=1,H=0)} \quad (20.49)$$

In general, we can write

$$p(\mathbf{x}_q|\mathbf{x}_v) = \frac{p(\mathbf{x}_q,\mathbf{x}_v)}{p(\mathbf{x}_v)} = \frac{\sum_{\mathbf{x}_h} p(\mathbf{x}_h,\mathbf{x}_q,\mathbf{x}_v)}{\sum_{\mathbf{x}_h}\sum_{\mathbf{x}'_q} p(\mathbf{x}_h,\mathbf{x}'_q,\mathbf{x}_v)} \quad (20.50)$$

The normalization constant in the denominator, $p(\mathbf{x}_v)$, is called the **probability of the evidence**.

See `variableElimination` for a simple Matlab implementation of this algorithm, which works for arbitrary graphs, and arbitrary discrete factors. But before you go too crazy, please read Section 20.3.2, which points out that VE can be exponentially slow in the worst case.

20.3.1 The generalized distributive law *

Abstractly, VE can be thought of as computing the following expression:

$$p(\mathbf{x}_q|\mathbf{x}_v) \propto \sum_{\mathbf{x}} \prod_c \psi_c(\mathbf{x}_c) \tag{20.51}$$

It is understood that the visible variables $\mathbf{x}_v$ are clamped, and not summed over. VE uses **non-serial dynamic programming** (Bertele and Brioschi 1972), caching intermediate results to avoid redundant computation.

However, there are other tasks we might like to solve for any given graphical model. For example, we might want the MAP estimate:

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} \prod_c \psi_c(\mathbf{x}_c) \tag{20.52}$$

Fortunately, essentially the same algorithm can also be used to solve this task: we just replace sum with max. (We also need a **traceback** step, which actually recovers the argmax, as opposed to just the value of $\max$; these details are explained in Section 17.4.4.)

In general, VE can be applied to any **commutative semi-ring**. This is a set K, together with two binary operations called "+" and "$\times$", which satisfy the following three axioms:

1. The operation "+" is associative and commutative, and there is an additive identity element called "0" such that $k + 0 = k$ for all $k \in K$.
2. The operation "$\times$" is associative and commutative, and there is a multiplicative identity element called "1" such that $k \times 1 = k$ for all $k \in K$.
3. The **distributive law** holds, i.e.,

$$(a \times b) + (a \times c) = a \times (b + c) \tag{20.53}$$

for all triples (a, b, c) from K.

This framework covers an extremely wide range of important applications, including constraint satisfaction problems (Bistarelli et al. 1997; Dechter 2003), the fast Fourier transform (Aji and McEliece 2000), etc. See Table 20.2 for some examples.

20.3.2 Computational complexity of VE

The running time of VE is clearly exponential in the size of the largest factor, since we have sum over all of the corresponding variables. Some of the factors come from the original model (and are thus unavoidable), but new factors are created in the process of summing out. For example,

Domain	+	×	Name
$[0, \infty)$	$(+, 0)$	$(\times, 1)$	sum-product
$[0, \infty)$	$(\max, 0)$	$(\times, 1)$	max-product
$(-\infty, \infty]$	$(\min, \infty)$	$(+, 0)$	min-sum
$\{T, F\}$	$(\vee, F)$	$(\wedge, T)$	Boolean satisfiability

Table 20.2 Some commutative semirings.

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \sum_I \psi_I(I)\psi_S(S,I) \underbrace{\sum_G \psi_G(G,I,D)\psi_L(L,)\psi_H(H,G,J)}_{\tau_1(I,D,L,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_I \psi_I(I)\psi_S(S,I)\tau_1(I,D,L,J,H)}_{\tau_2(D,L,S,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_2(D,L,S,J,H)}_{\tau_3(D,L,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \sum_H \underbrace{\sum_L \tau_3(D,L,J,H)}_{\tau_4(D,J,H)}$$

$$\sum_D \sum_C \psi_D(D,C) \underbrace{\sum_H \tau_4(D,J,H)}_{\tau_5(D,J)}$$

$$\sum_D \underbrace{\sum_C \psi_D(D,C)\tau_5(D,J)}_{\tau_6(D,J)}$$

$$\underbrace{\sum_D \tau_6(D,J)}_{\tau_7(J)}$$

Table 20.3 Eliminating variables from Figure 20.3 in the order G, I, S, L, H, C, D.

in Equation 20.47, we created a factor involving G, I and S; but these nodes were not originally present together in any factor.

The order in which we perform the summation is known as the **elimination order**. This can have a large impact on the size of the intermediate factors that are created. For example, consider the ordering in Table 20.1: the largest created factor (beyond the original ones in the model) has size 3, corresponding to $\tau_5(J, L, S)$. Now consider the ordering in Table 20.3: now the largest factors are $\tau_1(I, D, L, J, H)$ and $\tau_2(D, L, S, J, H)$, which are much bigger.

We can determine the size of the largest factor graphically, without worrying about the actual numerical values of the factors. When we eliminate a variable X_t, we connect it to all variables

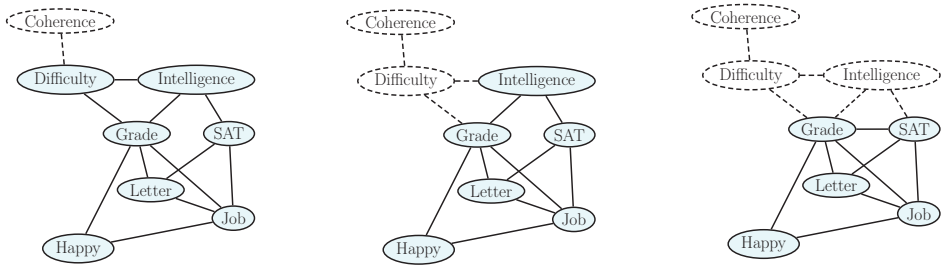

Figure 20.4 Example of the elimination process, in the order C, D, I, etc. When we eliminate I (figure c), we add a fill-in edge between G and S, since they are not connected. Based on Figure 9.10 of (Koller and Friedman 2009).

that share a factor with X_t (to reflect the new temporary factor τ'_t). The edges created by this process are called **fill-in edges**. For example, Figure 20.4 shows the fill-in edges introduced when we eliminate in the order $C, D, I, \ldots$. The first two steps do not introduce any fill-ins, but when we eliminate I, we connect G and S, since they co-occur in Equation 20.48.

Let $G(\prec)$ be the (undirected) graph induced by applying variable elimination to G using elimination ordering $\prec$. The temporary factors generated by VE correspond to maximal cliques in the graph $G(\prec)$. For example, with ordering (C, D, I, H, G, S, L), the maximal cliques are as follows:

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\} \tag{20.54}$$

It is clear that the time complexity of VE is

$$\prod_{c \in \mathcal{C}(G(\prec))} K^{|c|} \tag{20.55}$$

where $\mathcal{C}$ are the cliques that are created, $|c|$ is the size of the clique c, and we assume for notational simplicity that all the variables have K states each.

Let us define the **induced width** of a graph given elimination ordering $\prec$, denoted $w(\prec)$, as the size of the largest factor (i.e., the largest clique in the induced graph) minus 1. Then it is easy to see that the complexity of VE with ordering $\prec$ is $O(K^{w(\prec)+1})$.

Obviously we would like to minimize the running time, and hence the induced width. Let us define the **treewidth** of a graph as the minimal induced width.

$$w \triangleq \min_{\prec} \max_{c \in G(\prec)} |c| - 1 \tag{20.56}$$

Then clearly the best possible running time for VE is $O(DK^{w+1})$. Unfortunately, one can show that for arbitrary graphs, finding an elimination ordering $\prec$ that minimizes $w(\prec)$ is NP-hard (Arnborg et al. 1987). In practice greedy search techniques are used to find reasonable orderings (Kjaerulff 1990), although people have tried other heuristic methods for discrete optimization,

such as genetic algorithms (Larranaga et al. 1997). It is also possible to derive approximate algorithms with provable performance guarantees (Amir 2010).

In some cases, the optimal elimination ordering is clear. For example, for chains, we should work forwards or backwards in time. For trees, we should work from the leaves to the root. These orderings do not introduce any fill-in edges, so $w = 1$. Consequently, inference in chains and trees takes $O(VK^2)$ time. This is one reason why Markov chains and Markov trees are so widely used.

Unfortunately, for other graphs, the treewidth is large. For example, for an $m \times n$ 2d lattice, the treewidth is $O(\min\{m, n\})$ (Lipton and Tarjan 1979). So VE on a 100×100 Ising model would take $O(2^{100})$ time.

Of course, just because VE is slow doesn't mean that there isn't some smarter algorithm out there. We discuss this issue in Section 20.5.

20.3.3 A weakness of VE

The main disadvantage of the variable elimination algorithm (apart from its exponential dependence on treewidth) is that it is inefficient if we want to compute multiple queries conditioned on the same evidence. For example, consider computing all the marginals in a chain-structured graphical model such as an HMM. We can easily compute the final marginal $p(x_T|\mathbf{v})$ by eliminating all the nodes x_1 to x_{T-1} in order. This is equivalent to the forwards algorithm, and takes $O(K^2T)$ time. But now suppose we want to compute $p(x_{T-1}|\mathbf{v})$. We have to run VE again, at a cost of $O(K^2T)$ time. So the total cost to compute all the marginals is $O(K^2T^2)$. However, we know that we can solve this problem in $O(K^2T)$ using forwards-backwards. The difference is that FB caches the messages computed on the forwards pass, so it can reuse them later.

The same argument holds for BP on trees. For example, consider the 4-node tree in Figure 20.5. We can compute $p(x_1|\mathbf{v})$ by eliminating $\mathbf{x}_{2:4}$; this is equivalent to sending messages up to x_1 (the messages correspond to the τ factors created by VE). Similarly we can compute $p(x_2|\mathbf{v})$, $p(x_3|\mathbf{v})$ and then $p(x_4|\mathbf{v})$. We see that some of the messages used to compute the marginal on one node can be re-used to compute the marginals on the other nodes. By storing the messages for later re-use, we can compute all the marginals in $O(DK^2)$ time. This is what the up-down (collect-distribute) algorithm on trees does.

The question is: how can we combine the efficiency of BP on trees with the generality of VE? The answer is given in Section 20.4.

20.4 The junction tree algorithm *

The **junction tree algorithm** or **JTA** generalizes BP from trees to arbitrary graphs. We sketch the basic idea below; for details, see e.g., (Koller and Friedman 2009).

20.4.1 Creating a junction tree

The basic idea behind the JTA is this. We first run the VE algorithm "symbolically", adding fill-in edges as we go, according to a given elimination ordering. The resulting graph will be a **chordal graph**, which means that every undirected cycle $X_1 - X_2 \cdots X_k - X_1$ of length $k \geq 4$ has a

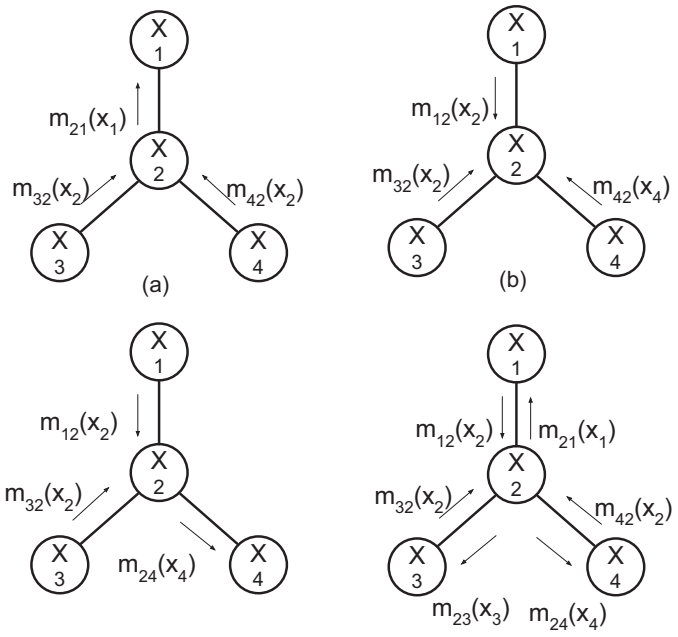

Figure 20.5 Sending multiple messages along a tree. (a) X_1 is root. (b) X_2 is root. (c) X_4 is root. (d) All of the messages needed to compute all singleton marginals. Based on Figure 4.3 of (Jordan 2007).

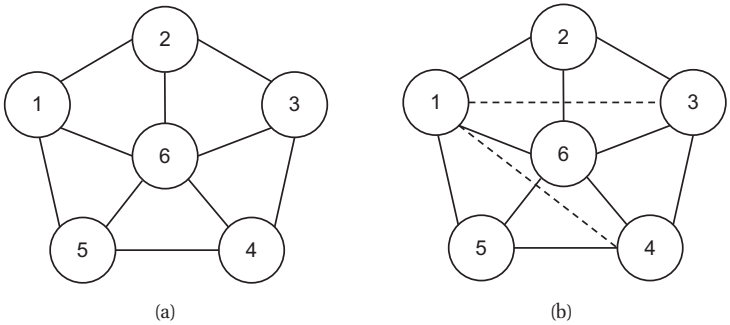

Figure 20.6 Left: this graph is not triangulated, despite appearances, since it contains a chordless 5-cycle 1-2-3-4-5-1. Right: one possible triangulation, by adding the 1-3 and 1-4 fill-in edges. Based on (Armstrong 2005, p46)

chord, i.e., an edge connects X_i, X_j for all non-adjacent nodes i,j in the cycle.[2]

Having created a chordal graph, we can extract its maximal cliques. In general, finding max cliques is computationally hard, but it turns out that it can be done efficiently from this special kind of graph. Figure 20.7(b) gives an example, where the max cliques are as follows:

$$\{C, D\}, \{G, I, D\}, \{G, S, I\}, \{G, J, S, L\}, \{H, G, J\} \tag{20.57}$$

Note that if the original graphical model was already chordal, the elimination process would not add any extra fill-in edges (assuming the optimal elimination ordering was used). We call such models **decomposable**, since they break into little pieces defined by the cliques.

It turns out that the cliques of a chordal graph can be arranged into a special kind of tree known as a **junction tree**. This enjoys the **running intersection property** (RIP), which means that any subset of nodes containing a given variable forms a connected component. Figure 20.7(c) gives an example of such a tree. We see that the node I occurs in two adjacent tree nodes, so they can share information about this variable. A similar situation holds for all the other variables.

One can show that if a tree that satisfies the running intersection property, then applying BP to this tree (as we explain below) will return the exact values of $p(\mathbf{x}_c|\mathbf{v})$ for each node c in the tree (i.e., clique in the induced graph). From this, we can easily extract the node and edge marginals, $p(x_t|\mathbf{v})$ and $p(x_s, x_t|\mathbf{v})$ from the original model, by marginalizing the clique distributions.[3]

20.4.2 Message passing on a junction tree

Having constructed a junction tree, we can use it for inference. The process is very similar to belief propagation on a tree. As in Section 20.2, there are two versions: the sum-product form, also known as the **Shafer-Shenoy** algorithm, named after (Shafer and Shenoy 1990); and the belief updating form (which involves division), also known as the **Hugin** (named after a company) or the **Lauritzen-Spiegelhalter** algorithm (named after (Lauritzen and Spiegelhalter 1988)). See (Lepar and Shenoy 1998) for a detailed comparison of these methods. Below we sketch how the Hugin algorithm works.

We assume the original model has the following form:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}(G)} \psi_c(\mathbf{x}_c) \tag{20.58}$$

where $\mathcal{C}(G)$ are the cliques of the original graph. On the other hand, the tree defines a distribution of the following form:

$$p(\mathbf{x}) = \frac{\prod_{c \in \mathcal{C}(T)} \psi_c(\mathbf{x}_c)}{\prod_{s \in \mathcal{S}(T)} \psi_s(\mathbf{x}_s)} \tag{20.59}$$

2. The largest loop in a chordal graph is length 3. Consequently chordal graphs are sometimes called **triangulated**. However, it is not enough for the graph to look like it is made of little triangles. For example, Figure 20.6(a) is not chordal, even though it is made of little triangles, since it contains the chordless 5-cycle 1-2-3-4-5-1.

3. If we want the joint distribution of some variables that are not in the same clique — a so-called **out-of-clique query** — we can adapt the technique described in Section 20.2.4.3 as follows: create a mega node containing the query variables and any other nuisance variables that lie on the path between them, multiply in messages onto the boundary of the mega node, and then marginalize out the internal nuisance variables. This internal marginalization may require the use of BP or VE. See (Koller and Friedman 2009) for details.

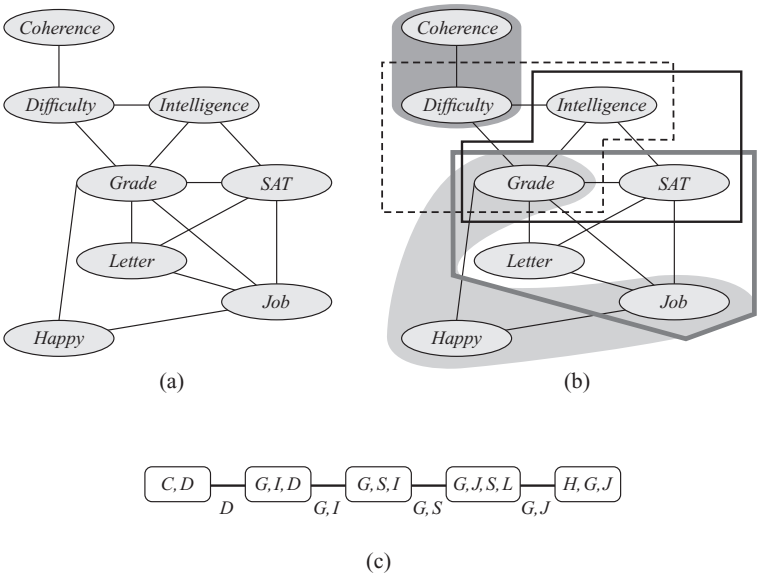

Figure 20.7 (a) The student graph with fill-in edges added. (b) The maximal cliques. (c) The junction tree. An edge between nodes s and t is labeled by the intersection of the sets on nodes s and t; this is called the **separating set**. From Figure 9.11 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

where $\mathcal{C}(T)$ are the nodes of the junction tree (which are the cliques of the chordal graph), and $\mathcal{S}(T)$ are the separators of the tree. To make these equal, we initialize by defining $\psi_s = 1$ for all separators and $\psi_c = 1$ for all cliques. Then, for each clique in the original model, $c \in \mathcal{C}(G)$, we find a clique in the tree $c' \in \mathcal{C}(T)$ which contains it, $c' \supseteq c$. We then multiply ψ_c onto $\psi_{c'}$ by computing $\psi_{c'} = \psi_{c'}\ \psi_c$. After doing this for all the cliques in the original graph, we have

$$\prod_{c \in \mathcal{C}(T)} \psi_c(\mathbf{x}_c) = \prod_{c \in \mathcal{C}(G)} \psi_c(\mathbf{x}_c) \tag{20.60}$$

As in Section 20.2.1, we now send messages from the leaves to the root and back, as sketched in Figure 20.1. In the upwards pass, also known as the **collect-to-root** phase, node i sends to its parent j the following message:

$$m_{i \to j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_i(C_i) \tag{20.61}$$

That is, we marginalize out the variables that node i "knows about" which are irrelevant to j, and then we send what is left over. Once a node has received messages from all its children, it updates its belief state using

$$\psi_i(C_i) \propto \psi_i(C_i) \prod_{j \in \text{ch}_i} m_{j \to i}(S_{ij}) \tag{20.62}$$

At the root, $\psi_r(C_r)$ represents $p(\mathbf{x}_{C_r}|\mathbf{v})$, which is the posterior over the nodes in clique C_r conditioned on all the evidence. Its normalization constant is $p(\mathbf{v})/Z_0$, where Z_0 is the normalization constant for the unconditional prior, $p(\mathbf{x})$. (We have $Z_0 = 1$ if the original model was a DGM.)

In the downwards pass, also known as the **distribute-from-root** phase, node i sends to its children j the following message:

$$m_{i\to j}(S_{ij}) = \frac{\sum_{C_i \setminus S_{ij}} \psi_i(C_i)}{m_{j\to i}(S_{ij})} \tag{20.63}$$

We divide out by what j sent to i to avoid double counting the evidence. This requires that we store the messages from the upwards pass. Once a node has received a top-down message from its parent, it can compute its final belief state using

$$\psi_j(C_j) \propto \psi_j(C_j) m_{i\to j}(S_{ij}) \tag{20.64}$$

An equivalent way to present this algorithm is based on storing the messages inside the separator potentials. So on the way up, sending from i to j we compute the separator potential

$$\psi^*_{ij}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_i(C_i) \tag{20.65}$$

and then update the recipient potential:

$$\psi^*_j(C_j) \propto \psi_j(C_j) \frac{\psi^*_{ij}(S_{ij})}{\psi_{ij}(S_{ij})} \tag{20.66}$$

(Recall that we initialize $\psi_{ij}(S_{ij}) = 1$.) This is sometimes called **passing a flow** from i to j. On the way down, from i to j, we compute the separator potential

$$\psi^{**}_{ij}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi^*_i(C_i) \tag{20.67}$$

and then update the recipient potential:

$$\psi^{**}_j(C_j) \propto \psi^*_j(C_j) \frac{\psi^{**}_{ij}(S_{ij})}{\psi^*_{ij}(S_{ij})} \tag{20.68}$$

This process is known as junction tree **calibration**. See Figure 20.1 for an illustration. Its correctness follows from the fact that each edge partitions the evidence into two distinct groups, plus the fact that the tree satisfies RIP, which ensures that no information is lost by only performing local computations.

20.4.2.1 Example: jtree algorithm on a chain

It is interesting to see what happens if we apply this process to a chain structured graph such as an HMM. A detailed discussion can be found in (Smyth et al. 1997), but the basic idea is this. The cliques are the edges, and the separators are the nodes, as shown in Figure 20.8. We initialize the potentials as follows: we set $\psi_s = 1$ for all the separators, we set $\psi_c(x_{t-1}, x_t) = p(x_t|x_{t-1})$ for clique $c = (X_{t-1}, X_t)$, and we set $\psi_c(x_t, y_t) = p(y_t|x_t)$ for clique $c = (X_t, Y_t)$.

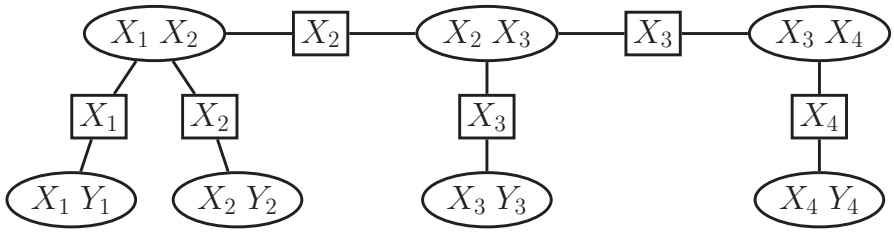

Figure 20.8 The junction tree derived from an HMM/SSM of length $T = 4$.

Next we send messages from left to right. Consider clique (X_{t-1}, X_t) with potential $p(X_t|X_{t-1})$. It receives a message from clique (X_{t-2}, X_{t-1}) via separator X_{t-1} of the form $\sum_{x_{t-2}} p(X_{t-2}, X_{t-1}|\mathbf{v}_{1:t-1}) = p(X_{t-1}|\mathbf{v}_{1:t-1})$. When combined with the clique potential, this becomes the two-slice predictive density

$$p(X_t|X_{t-1})p(X_{t-1}|\mathbf{v}_{1:t-1}) = p(X_{t-1}, X_t|\mathbf{v}_{1:t-1} \tag{20.69}$$

The clique (X_{t-1}, X_t) also receives a message from (X_t, Y_t) via separator X_t of the form $p(y_t|X_t)$, which corresponds to its local evidence. When combined with the updated clique potential, this becomes the two-slice filtered posterior

$$p(X_{t-1}, X_t|\mathbf{v}_{1:t-1})p(\mathbf{v}_t|X_t) = p(X_{t-1}, X_t|\mathbf{v}_{1:t}) \tag{20.70}$$

Thus the messages in the forwards pass are the filtered belief states α_t, and the clique potentials are the two-slice distributions. In the backwards pass, the messages are the update factors $\frac{\gamma_t}{\alpha_t}$, where $\gamma_t(k) = p(x_t = k|\mathbf{v}_{1:T})$ and $\alpha_t(k) = p(x_t = k|\mathbf{v}_{1:t})$. By multiplying by this message, we "swap out" the old α_t message and "swap in" the new γ_t message. We see that the backwards pass involves working with posterior beliefs, not conditional likelihoods. See Section 18.3.2.3 for further discussion of this difference.

20.4.3 Computational complexity of JTA

If all nodes are discrete with K states each, it is clear that the JTA takes $O(|\mathcal{C}|K^{w+1})$ time and space, where $|\mathcal{C}|$ is the number of cliques and w is the treewidth of the graph, i.e., the size of the largest clique minus 1. Unfortunately, choosing a triangulation so as to minimize the treewidth is NP-hard, as explained in Section 20.3.2.

The JTA can be modified to handle the case of Gaussian graphical models. The graph-theoretic steps remain unchanged. Only the message computation differs. We just need to define how to multiply, divide, and marginalize Gaussian potential functions. This is most easily done in information form. See e.g., (Lauritzen 1992; Murphy 1998; Cemgil 2001) for the details. The algorithm takes $O(|\mathcal{C}|w^3)$ time and $O(|\mathcal{C}|w^2)$ space. When applied to a chain structured graph, the algorithm is equivalent to the Kalman smoother in Section 18.3.2.

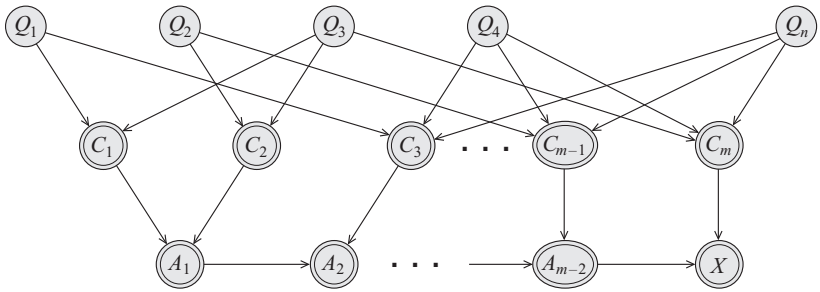

Figure 20.9 Encoding a 3-SAT problem on n variables and m clauses as a DGM. The Q_s variables are binary random variables. The C_t variables are deterministic functions of the Q_s's, and compute the truth value of each clause. The A_t nodes are a chain of AND gates, to ensure that the CPT for the final x node has bounded size. The double rings denote nodes with deterministic CPDs. Source: Figure 9.1 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

20.4.4 JTA generalizations *

We have seen how to use the JTA algorithm to compute posterior marginals in a graphical model. There are several possible generalizations of this algorithm, some of which we mention below. All of these exploit graph decomposition in some form or other. They only differ in terms of how they define/ compute messages and "beliefs". The key requirement is that the operators which compute messages form a commutative semiring (see Section 20.3.1).

- Computing the MAP estimate. We just replace the sum-product with max-product in the collect phase, and use traceback in the distribute phase, as in the Viterbi algorithm (Section 17.4.4). See (Dawid 1992) for details.
- Computing the N-most probable configurations (Nilsson 1998).
- Computing posterior samples. The collect pass is the same as usual, but in the distribute pass, we sample variables given the values higher up in the tree, thus generalizing forwards-filtering backwards-sampling for HMMs described in Section 17.4.5. See (Dawid 1992) for details.
- Solving constraint satisfaction problems (Dechter 2003).
- Solving logical reasoning problems (Amir and McIlraith 2005).

20.5 Computational intractability of exact inference in the worst case

As we saw in Sections 20.3.2 and 20.4.3, VE and JTA take time that is exponential in the treewidth of a graph. Since the treewidth can be O(number of nodes) in the worst case, this means these algorithms can be exponential in the problem size.

Of course, just because VE and JTA are slow doesn't mean that there isn't some smarter algorithm out there. Unfortunately, this seems unlikely, since it is easy to show that exact inference is NP-hard (Dagum and Luby 1993). The proof is a simple reduction from the satisfiability prob-

Method	Restriction	Section
Forwards-backwards	Chains, D or LG	Section 17.4.3
Belief propagation	Trees, D or LG	Section 20.2
Variable elimination	Low treewidth, D or LG, single query	Section 20.3
Junction tree algorithm	Low treewidth, D or LG	Section 20.4
Loopy belief propagation	Approximate, D or LG	Section 22.2
Convex belief propagation	Approximate, D or LG	Section 22.4.2
Mean field	Approximate, C-E	Section 21.3
Gibbs sampling	Approximate	Section 24.2

Table 20.4 Summary of some methods that can be used for inference in graphical models. "D" means that all the hidden variables must be discrete. "L-G" means that all the factors must be linear-Gaussian. The term "single query" refers to the restriction that VE only computes one marginal $p(\mathbf{x}_q|\mathbf{x}_v)$ at a time. See Section 20.3.3 for a discussion of this point. "C-E" stands for "conjugate exponential"; this means that variational mean field only applies to models where the likelihood is in the exponential family, and the prior is conjugate. This includes the D and LG case, but many others as well, as we will see in Section 21.5.

lem. In particular, note that we can encode any 3-SAT problem[4] as a DGM with deterministic links, as shown in Figure 20.9. We clamp the final node, x, to be on, and we arrange the CPTs so that $p(x = 1) > 0$ iff there a satisfying assignment. Computing any posterior marginal requires evaluating the normalization constant $p(x = 1)$, which represents the probability of the evidence, so inference in this model implicitly solves the SAT problem.

In fact, exact inference is #P-hard (Roth 1996), which is even harder than NP-hard. (See e.g., (Arora and Barak 2009) for definitions of these terms.) The intuitive reason for this is that to compute the normalizing constant Z, we have to *count* how many satisfying assignments there are. By contrast, MAP estimation is provably easier for some model classes (Greig et al. 1989), since, intuitively speaking, it only requires finding one satisfying assignment, not counting all of them.

20.5.1 Approximate inference

Many popular probabilistic models support efficient exact inference, since they are based on chains, trees or low treewidth graphs. But there are many other models for which exact inference is intractable. In fact, even simple two node models of the form $\boldsymbol{\theta} \rightarrow \mathbf{x}$ may not support exact inference if the prior on $\boldsymbol{\theta}$ is not conjugate to the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$.[5]

Therefore we will need to turn to **approximate inference** methods. See Table 20.4 for a summary of coming attractions. For the most part, these methods do not come with any guarantee as to their accuracy or running time. Theoretical computer scientists would therefore describe them as **heuristics** rather than approximation algorithms. In fact, one can prove that

4. A 3-SAT problem is a logical expression of the form $(Q_1 \wedge Q_2 \wedge \neg Q_3) \vee (Q_1 \wedge \neg Q_4 \wedge Q_5) \cdots$, where the Q_i are binary variables, and each clause consists of the conjunction of three variables (or their negation). The goal is to find a satisfying assignment, which is a set of values for the Q_i variables such that the expression evaluates to true.

5. For discrete random variables, conjugacy is not a concern, since discrete distributions are always closed under conditioning and marginalization. Consequently, graph-theoretic considerations are of more importance when discussing inference in models with discrete hidden states.

it is not possible to construct polynomial time approximation schemes for inference in general discrete GMs (Dagum and Luby 1993; Roth 1996). Fortunately, we will see that for many of these heuristic methods often perform well in practice.

Exercises

Exercise 20.1 Variable elimination

Consider the MRF in Figure 10.14(b).

a. Suppose we want to compute the partition function using the elimination ordering $\prec = (1, 2, 3, 4, 5, 6)$, i.e.,

$$\sum_{x_6}\sum_{x_5}\sum_{x_4}\sum_{x_3}\sum_{x_2}\sum_{x_1}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{34}(x_3,x_4)\psi_{45}(x_4,x_5)\psi_{56}(x_5,x_6) \quad (20.71)$$

If we use the variable elimination algorithm, we will create new intermediate factors. What is the largest intermediate factor?

b. Add an edge to the original MRF between every pair of variables that end up in the same factor. (These are called fill in edges.) Draw the resulting MRF. What is the size of the largest maximal clique in this graph?

c. Now consider elimination ordering $\prec = (4, 1, 2, 3, 5, 6)$, i.e.,

$$\sum_{x_6}\sum_{x_5}\sum_{x_3}\sum_{x_2}\sum_{x_1}\sum_{x_4}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{34}(x_3,x_4)\psi_{45}(x_4,x_5)\psi_{56}(x_5,x_6) \quad (20.72)$$

If we use the variable elimination algorithm, we will create new intermediate factors. What is the largest intermediate factor?

d. Add an edge to the original MRF between every pair of variables that end up in the same factor. (These are called fill in edges.) Draw the resulting MRF. What is the size of the largest maximal clique in this graph?

Exercise 20.2 Gaussian times Gaussian is Gaussian

Prove Equation 20.17. Hint: use completing the square.

Exercise 20.3 Message passing on a tree

Consider the DGM in Figure 20.10 which represents the following fictitious biological model. Each G_i represents the genotype of a person: $G_i = 1$ if they have a healthy gene and $G_i = 2$ if they have an unhealthy gene. G_2 and G_3 may inherit the unhealthy gene from their parent G_1. $X_i \in \mathbb{R}$ is a continuous measure of blood pressure, which is low if you are healthy and high if you are unhealthy. We define the CPDs as follows

$$p(G_1) = [0.5, 0.5] \quad (20.73)$$

$$p(G_2|G_1) = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \quad (20.74)$$

$$p(G_3|G_1) = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \quad (20.75)$$

$$p(X_i|G_i = 1) = \mathcal{N}(X_i|\mu = 50, \sigma^2 = 10) \quad (20.76)$$

$$p(X_i|G_i = 2) = \mathcal{N}(X_i|\mu = 60, \sigma^2 = 10) \quad (20.77)$$

The meaning of the matrix for $p(G_2|G_1)$ is that $p(G_2 = 1|G_1 = 1) = 0.9$, $p(G_2 = 1|G_1 = 2) = 0.1$, etc.

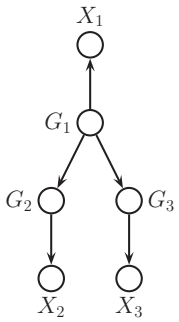

Figure 20.10 A simple DAG representing inherited diseases.

a. Suppose you observe $X_2 = 50$, and X_1 is unobserved. What is the posterior belief on G_1, i.e., $p(G_1|X_2 = 50)$?

b. Now suppose you observe $X_2 = 50$ amd $X_3 = 50$. What is $p(G_1|X_2, X_3)$? Explain your answer intuitively.

c. Now suppose $X_2 = 60$, $X_3 = 60$. What is $p(G_1|X_2, X_3)$? Explain your answer intuitively.

d. Now suppose $X_2 = 50$, $X_3 = 60$. What is $p(G_1|X_1, X_2)$? Explain your answer intuitively.

Exercise 20.4 Inference in 2D lattice MRFs

Consider an MRF with a 2D $m \times n$ lattice graph structure, so each hidden node, X_{ij}, is connected to its 4 nearest neighbors, as in an Ising model. In addition, each hidden node has its own local evidence, Y_{ij}. Assume all hidden nodes have $K > 2$ states. In general, exact inference in such models is intractable, because the maximum cliques of the corresponding triangulated graph have size $O(\max\{m, n\})$. Suppose $m \ll n$ i.e., the lattice is short and fat.

a. How can one *efficiently* perform exact inference (using a deterministic algorithm) in such models? (By exact inference, I mean computing marginal probabilities $P(X_{ij}|\vec{y})$ exactly, where $\vec{y}$ is all the evidence.) Give a *brief* description of your method.

b. What is the asymptotic complexity (running time) of your algorithm?

c. Now suppose the lattice is large and square, so $m = n$, but all hidden states are binary (ie $K = 2$). In this case, how can one efficiently exactly compute (using a deterministic algorithm) the MAP estimate $\arg\max_x P(x|y)$, where x is the joint assignment to all hidden nodes?