

Figure 8.43 (a) A directed polytree. (b) The result of converting the polytree into an undirected graph showing the creation of loops. (c) The result of converting the polytree into a factor graph, which retains the tree structure.

precise form of the factorization. Figure 8.45 shows an example of a fully connected undirected graph along with two different factor graphs. In (b), the joint distribution is given by a general form $p(\mathbf{x}) = f(x_1, x_2, x_3)$, whereas in (c), it is given by the more specific factorization $p(\mathbf{x}) = f_a(x_1, x_2)f_b(x_1, x_3)f_c(x_2, x_3)$. It should be emphasized that the factorization in (c) does not correspond to any conditional independence properties.

8.4.4 The sum-product algorithm

We shall now make use of the factor graph framework to derive a powerful class of efficient, exact inference algorithms that are applicable to tree-structured graphs. Here we shall focus on the problem of evaluating local marginals over nodes or subsets of nodes, which will lead us to the *sum-product* algorithm. Later we shall modify the technique to allow the most probable state to be found, giving rise to the *max-sum* algorithm.

Also we shall suppose that all of the variables in the model are discrete, and so marginalization corresponds to performing sums. The framework, however, is equally applicable to linear-Gaussian models in which case marginalization involves integration, and we shall consider an example of this in detail when we discuss linear dynamical systems.

Section 13.3

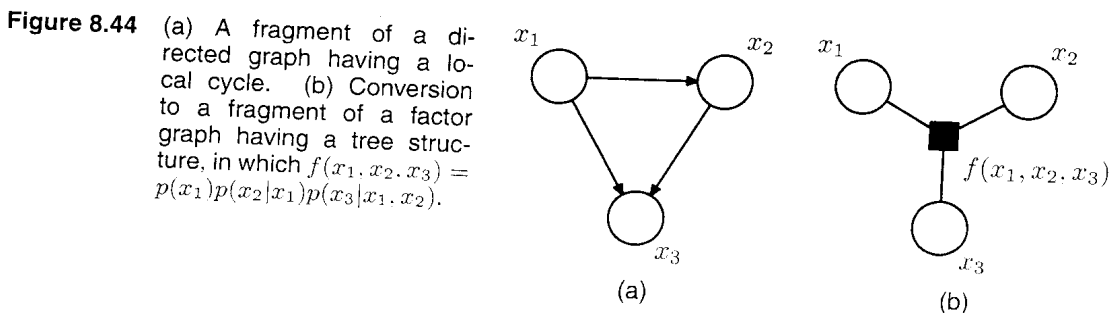


Figure 8.44 (a) A fragment of a directed graph having a local cycle. (b) Conversion to a fragment of a factor graph having a tree structure, in which $f(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)$.

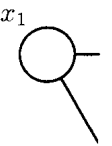


Figure 8.45 to the undirect

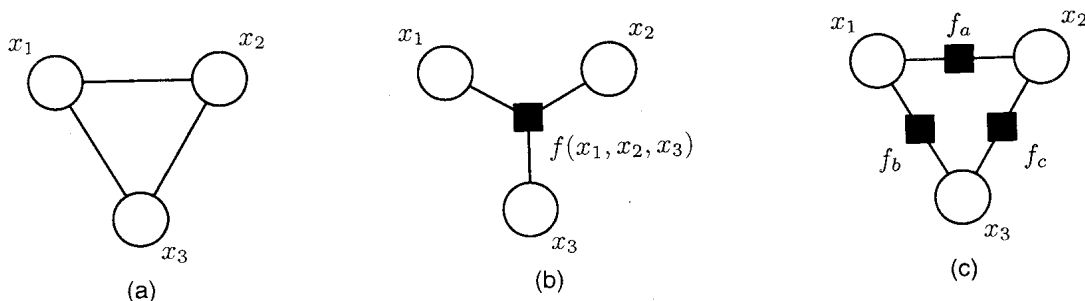


Figure 8.45 (a) A fully connected undirected graph. (b) and (c) Two factor graphs each of which corresponds to the undirected graph in (a).

There is an algorithm for exact inference on directed graphs without loops known as *belief propagation* (Pearl, 1988; Lauritzen and Spiegelhalter, 1988), and is equivalent to a special case of the sum-product algorithm. Here we shall consider only the sum-product algorithm because it is simpler to derive and to apply, as well as being more general.

We shall assume that the original graph is an undirected tree or a directed tree or polytree, so that the corresponding factor graph has a tree structure. We first convert the original graph into a factor graph so that we can deal with both directed and undirected models using the same framework. Our goal is to exploit the structure of the graph to achieve two things: (i) to obtain an efficient, exact inference algorithm for finding marginals; (ii) in situations where several marginals are required to allow computations to be shared efficiently.

We begin by considering the problem of finding the marginal $p(x)$ for particular variable node x . For the moment, we shall suppose that all of the variables are hidden. Later we shall see how to modify the algorithm to incorporate evidence corresponding to observed variables. By definition, the marginal is obtained by summing the joint distribution over all variables except x so that

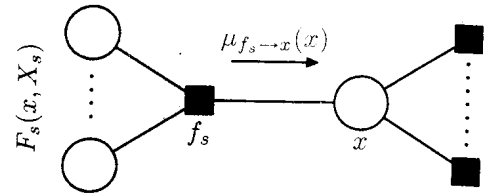
$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x}) \quad (8.61)$$

where $\mathbf{x} \setminus x$ denotes the set of variables in \mathbf{x} with variable x omitted. The idea is to substitute for $p(\mathbf{x})$ using the factor graph expression (8.59) and then interchange summations and products in order to obtain an efficient algorithm. Consider the fragment of graph shown in Figure 8.46 in which we see that the tree structure of the graph allows us to partition the factors in the joint distribution into groups, with one group associated with each of the factor nodes that is a neighbour of the variable node x . We see that the joint distribution can be written as a product of the form

$$p(\mathbf{x}) = \prod_{s \in \text{ne}(x)} F_s(x, X_s) \quad (8.62)$$

$\text{ne}(x)$ denotes the set of factor nodes that are neighbours of x , and X_s denotes the set of all variables in the subtree connected to the variable node x via the factor node

Figure 8.46 A fragment of a factor graph illustrating the evaluation of the marginal $p(x)$.



f_s , and $F_s(x, X_s)$ represents the product of all the factors in the group associated with factor f_s .

Substituting (8.62) into (8.61) and interchanging the sums and products, we obtain

$$\begin{aligned} p(x) &= \prod_{s \in \text{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] \\ &= \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x). \end{aligned} \quad (8.63)$$

Here we have introduced a set of functions $\mu_{f_s \rightarrow x}(x)$, defined by

$$\mu_{f_s \rightarrow x}(x) \triangleq \sum_{X_s} F_s(x, X_s) \quad (8.64)$$

which can be viewed as *messages* from the factor nodes f_s to the variable node x . We see that the required marginal $p(x)$ is given by the product of all the incoming messages arriving at node x .

In order to evaluate these messages, we again turn to Figure 8.46 and note that each factor $F_s(x, X_s)$ is described by a factor (sub-)graph and so can itself be factorized. In particular, we can write

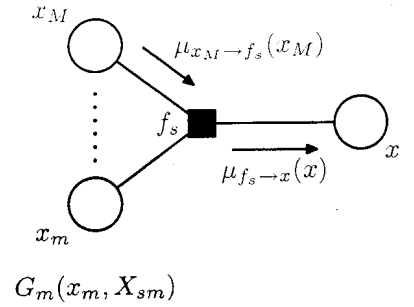
$$F_s(x, X_s) = f_s(x, x_1, \dots, x_M) G_1(x_1, X_{s1}) \dots G_M(x_M, X_{sM}) \quad (8.65)$$

where, for convenience, we have denoted the variables associated with factor f_s , in addition to x , by x_1, \dots, x_M . This factorization is illustrated in Figure 8.47. Note that the set of variables $\{x, x_1, \dots, x_M\}$ is the set of variables on which the factor f_s depends, and so it can also be denoted \mathbf{x}_s , using the notation of (8.59).

Substituting (8.65) into (8.64) we obtain

$$\begin{aligned} \mu_{f_s \rightarrow x}(x) &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \left[\sum_{X_{xm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m) \end{aligned} \quad (8.66)$$

Figure 8.47 Illustration of the factorization of the subgraph associated with factor node f_s .



where $ne(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s , and $ne(f_s) \setminus x$ denotes the same set but with node x removed. Here we have defined the following messages from variable nodes to factor nodes

$$\mu_{x_m \rightarrow f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm}). \tag{8.67}$$

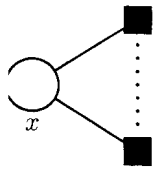
We have therefore introduced two distinct kinds of message, those that go from factor nodes to variable nodes denoted $\mu_{f \rightarrow x}(x)$, and those that go from variable nodes to factor nodes denoted $\mu_{x \rightarrow f}(x)$. In each case, we see that messages passed along a link are always a function of the variable associated with the variable node that link connects to.

The result (8.66) says that to evaluate the message sent by a factor node to a variable node along the link connecting them, take the product of the incoming messages along all other links coming into the factor node, multiply by the factor associated with that node, and then marginalize over all of the variables associated with the incoming messages. This is illustrated in Figure 8.47. It is important to note that a factor node can send a message to a variable node once it has received incoming messages from all other neighbouring variable nodes.

Finally, we derive an expression for evaluating the messages from variable nodes to factor nodes, again by making use of the (sub-)graph factorization. From Figure 8.48, we see that term $G_m(x_m, X_{sm})$ associated with node x_m is given by a product of terms $F_l(x_m, X_{ml})$ each associated with one of the factor nodes f_l that is linked to node x_m (excluding node f_s), so that

$$G_m(x_m, X_{sm}) = \prod_{l \in ne(x_m) \setminus f_s} F_l(x_m, X_{ml}) \tag{8.68}$$

where the product is taken over all neighbours of node x_m except for node f_s . Note that each of the factors $F_l(x_m, X_{ml})$ represents a subtree of the original graph of precisely the same kind as introduced in (8.62). Substituting (8.68) into (8.67), we



up associated
ducts, we ob-

(8.63)

(8.64)

able node x .
he incoming

nd note that
tself be fac-

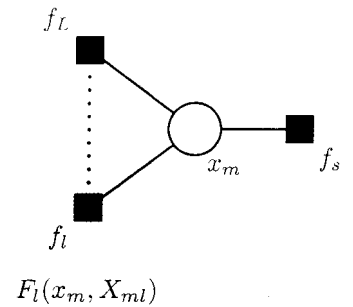
(8.65)

factor f_x , in
: 8.47. Note
h the factor
)

$m, X_{sm})$]

(8.66)

Figure 8.48 Illustration of the evaluation of the message sent by a variable node to an adjacent factor node.



then obtain

$$\begin{aligned} \mu_{x_m \rightarrow f_s}(x_m) &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] \\ &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m) \end{aligned} \quad (8.69)$$

where we have used the definition (8.64) of the messages passed from factor nodes to variable nodes. Thus to evaluate the message sent by a variable node to an adjacent factor node along the connecting link, we simply take the product of the incoming messages along all of the other links. Note that any variable node that has only two neighbours performs no computation but simply passes messages through unchanged. Also, we note that a variable node can send a message to a factor node once it has received incoming messages from all other neighbouring factor nodes.

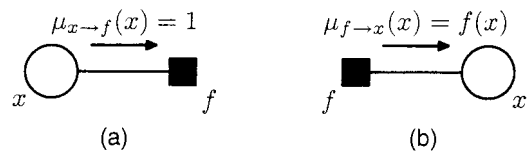
Recall that our goal is to calculate the marginal for variable node x , and that this marginal is given by the product of incoming messages along all of the links arriving at that node. Each of these messages can be computed recursively in terms of other messages. In order to start this recursion, we can view the node x as the root of the tree and begin at the leaf nodes. From the definition (8.69), we see that if a leaf node is a variable node, then the message that it sends along its one and only link is given by

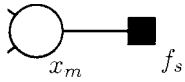
$$\mu_{x \rightarrow f}(x) = 1 \quad (8.70)$$

as illustrated in Figure 8.49(a). Similarly, if the leaf node is a factor node, we see from (8.66) that the message sent should take the form

$$\mu_{f \rightarrow x}(x) = f(x) \quad (8.71)$$

Figure 8.49 The sum-product algorithm begins with messages sent by the leaf nodes, which depend on whether the leaf node is (a) a variable node, or (b) a factor node.





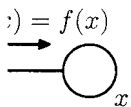
(8.69)

... factor nodes to
... to an adjacent
... of the incoming
... that has only
... ges through un-
... to a factor node
... factor nodes.
... e x , and that this
... he links arriving
... n terms of other
... s the root of the
... at if a leaf node
... nly link is given

(8.70)

... or node, we see

(8.71)



(b)

as illustrated in Figure 8.49(b).

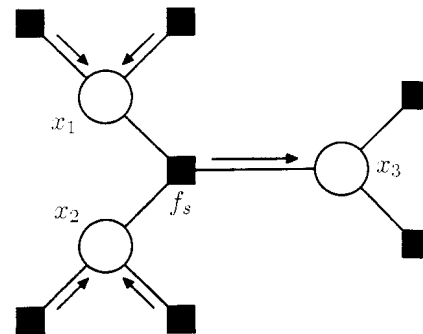
At this point, it is worth pausing to summarize the particular version of the sum-product algorithm obtained so far for evaluating the marginal $p(x)$. We start by viewing the variable node x as the root of the factor graph and initiating messages at the leaves of the graph using (8.70) and (8.71). The message passing steps (8.66) and (8.69) are then applied recursively until messages have been propagated along every link, and the root node has received messages from all of its neighbours. Each node can send a message towards the root once it has received messages from all of its other neighbours. Once the root node has received messages from all of its neighbours, the required marginal can be evaluated using (8.63). We shall illustrate this process shortly.

To see that each node will always receive enough messages to be able to send out a message, we can use a simple inductive argument as follows. Clearly, for a graph comprising a variable root node connected directly to several factor leaf nodes, the algorithm trivially involves sending messages of the form (8.71) directly from the leaves to the root. Now imagine building up a general graph by adding nodes one at a time, and suppose that for some particular graph we have a valid algorithm. When one more (variable or factor) node is added, it can be connected only by a single link because the overall graph must remain a tree, and so the new node will be a leaf node. It therefore sends a message to the node to which it is linked, which in turn will therefore receive all the messages it requires in order to send its own message towards the root, and so again we have a valid algorithm, thereby completing the proof.

Now suppose we wish to find the marginals for every variable node in the graph. This could be done by simply running the above algorithm afresh for each such node. However, this would be very wasteful as many of the required computations would be repeated. We can obtain a much more efficient procedure by 'overlying' these multiple message passing algorithms to obtain the general sum-product algorithm as follows. Arbitrarily pick any (variable or factor) node and designate it as the root. Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours. It can therefore send out messages to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on. In this way, messages are passed outwards from the root all the way to the leaves. By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours. Again a simple inductive argument can be used to verify the validity of this message passing protocol. Because every variable node will have received messages from all of its neighbours, we can readily calculate the marginal distribution for every variable in the graph. The number of messages that have to be computed is given by twice the number of links in the graph and so involves only twice the computation involved in finding a single marginal. By comparison, if we had run the sum-product algorithm separately for each node, the amount of computation would grow quadratically with the size of the graph. Note that this algorithm is in fact independent of which node was designated as the root,

Exercise 8.20

Figure 8.50 The sum-product algorithm can be viewed purely in terms of messages sent out by factor nodes to other factor nodes. In this example, the outgoing message shown by the blue arrow is obtained by taking the product of all the incoming messages shown by green arrows, multiplying by the factor f_s , and marginalizing over the variables x_1 and x_2 .



and indeed the notion of one node having a special status was introduced only as a convenient way to explain the message passing protocol.

Next suppose we wish to find the marginal distributions $p(\mathbf{x}_s)$ associated with the sets of variables belonging to each of the factors. By a similar argument to that used above, it is easy to see that the marginal associated with a factor is given by the product of messages arriving at the factor node and the local factor at that node

Exercise 8.21

$$p(\mathbf{x}_s) = f_s(\mathbf{x}_s) \prod_{i \in \text{ne}(f_s)} \mu_{x_i \rightarrow f_s}(x_i) \quad (8.72)$$

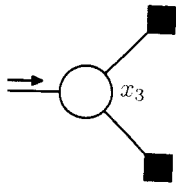
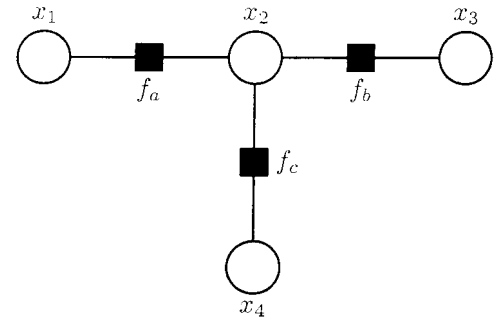
in complete analogy with the marginals at the variable nodes. If the factors are parameterized functions and we wish to learn the values of the parameters using the EM algorithm, then these marginals are precisely the quantities we will need to calculate in the E step, as we shall see in detail when we discuss the hidden Markov model in Chapter 13.

The message sent by a variable node to a factor node, as we have seen, is simply the product of the incoming messages on other links. We can if we wish view the sum-product algorithm in a slightly different form by eliminating messages from variable nodes to factor nodes and simply considering messages that are sent out by factor nodes. This is most easily seen by considering the example in Figure 8.50.

So far, we have rather neglected the issue of normalization. If the factor graph was derived from a directed graph, then the joint distribution is already correctly normalized, and so the marginals obtained by the sum-product algorithm will similarly be normalized correctly. However, if we started from an undirected graph, then in general there will be an unknown normalization coefficient $1/Z$. As with the simple chain example of Figure 8.38, this is easily handled by working with an unnormalized version $\tilde{p}(\mathbf{x})$ of the joint distribution, where $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z$. We first run the sum-product algorithm to find the corresponding unnormalized marginals $\tilde{p}(x_i)$. The coefficient $1/Z$ is then easily obtained by normalizing any one of these marginals, and this is computationally efficient because the normalization is done over a single variable rather than over the entire set of variables as would be required to normalize $\tilde{p}(\mathbf{x})$ directly.

At this point, it may be helpful to consider a simple example to illustrate the operation of the sum-product algorithm. Figure 8.51 shows a simple 4-node factor

Figure 8.51 A simple factor graph used to illustrate the sum-product algorithm.



duced only as a
associated with
argument to that
is given by the
that node

$$(8.72)$$

he factors are
ameters using
e will need to
idden Markov

een, is simply
wish view the
essages from
re sent out by
igure 8.50.

Factor graph
correctly nor-
will similarly
graph, then in
ith the simple
an unnormal-
first run the
ls $\tilde{p}(x_i)$. The
se marginals,
over a single
to normalize

illustrate the
l-node factor

graph whose unnormalized joint distribution is given by

$$\tilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4). \quad (8.73)$$

In order to apply the sum-product algorithm to this graph, let us designate node x_3 as the root, in which case there are two leaf nodes x_1 and x_4 . Starting with the leaf nodes, we then have the following sequence of six messages

$$\mu_{x_1 \rightarrow f_a}(x_1) = 1 \quad (8.74)$$

$$\mu_{f_a \rightarrow x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2) \quad (8.75)$$

$$\mu_{x_4 \rightarrow f_c}(x_4) = 1 \quad (8.76)$$

$$\mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4) \quad (8.77)$$

$$\mu_{x_2 \rightarrow f_b}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \quad (8.78)$$

$$\mu_{f_b \rightarrow x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}. \quad (8.79)$$

The direction of flow of these messages is illustrated in Figure 8.52. Once this message propagation is complete, we can then propagate messages from the root node out to the leaf nodes, and these are given by

$$\mu_{x_3 \rightarrow f_b}(x_3) = 1 \quad (8.80)$$

$$\mu_{f_b \rightarrow x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3) \quad (8.81)$$

$$\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \quad (8.82)$$

$$\mu_{f_a \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_2) \quad (8.83)$$

$$\mu_{x_2 \rightarrow f_c}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2) \quad (8.84)$$

$$\mu_{f_c \rightarrow x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \rightarrow f_c}(x_2). \quad (8.85)$$

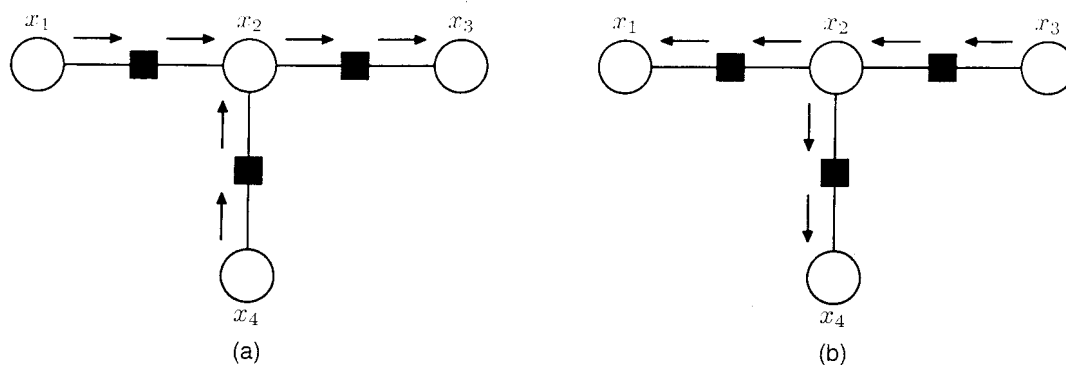


Figure 8.52 Flow of messages for the sum-product algorithm applied to the example graph in Figure 8.51. (a) From the leaf nodes x_1 and x_4 towards the root node x_3 . (b) From the root node towards the leaf nodes.

One message has now passed in each direction across each link, and we can now evaluate the marginals. As a simple check, let us verify that the marginal $p(x_2)$ is given by the correct expression. Using (8.63) and substituting for the messages using the above results, we have

$$\begin{aligned}
 \tilde{p}(x_2) &= \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \\
 &= \left[\sum_{x_1} f_a(x_1, x_2) \right] \left[\sum_{x_3} f_b(x_2, x_3) \right] \left[\sum_{x_4} f_c(x_2, x_4) \right] \\
 &= \sum_{x_1} \sum_{x_2} \sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\
 &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \tilde{p}(\mathbf{x})
 \end{aligned} \tag{8.86}$$

as required.

So far, we have assumed that all of the variables in the graph are hidden. In most practical applications, a subset of the variables will be observed, and we wish to calculate posterior distributions conditioned on these observations. Observed nodes are easily handled within the sum-product algorithm as follows. Suppose we partition \mathbf{x} into hidden variables \mathbf{h} and observed variables \mathbf{v} , and that the observed value of \mathbf{v} is denoted $\hat{\mathbf{v}}$. Then we simply multiply the joint distribution $p(\mathbf{x})$ by $\prod_i I(v_i, \hat{v}_i)$, where $I(v, \hat{v}) = 1$ if $v = \hat{v}$ and $I(v, \hat{v}) = 0$ otherwise. This product corresponds to $p(\mathbf{h}, \mathbf{v} = \hat{\mathbf{v}})$ and hence is an unnormalized version of $p(\mathbf{h} | \mathbf{v} = \hat{\mathbf{v}})$. By running the sum-product algorithm, we can efficiently calculate the posterior marginals $p(h_i | \mathbf{v} = \hat{\mathbf{v}})$ up to a normalization coefficient whose value can be found efficiently using a local computation. Any summations over variables in \mathbf{v} then collapse into a single term.

We have assumed throughout this section that we are dealing with discrete variables. However, there is nothing specific to discrete variables either in the graphical framework or in the probabilistic construction of the sum-product algorithm. For

Table 8.1 Example of a joint distribution over two binary variables for which the maximum of the joint distribution occurs for different variable values compared to the maxima of the two marginals.

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

Section 13.3

continuous variables the summations are simply replaced by integrations. We shall give an example of the sum-product algorithm applied to a graph of linear-Gaussian variables when we consider linear dynamical systems.

8.4.5 The max-sum algorithm

The sum-product algorithm allows us to take a joint distribution $p(\mathbf{x})$ expressed as a factor graph and efficiently find marginals over the component variables. Two other common tasks are to find a setting of the variables that has the largest probability and to find the value of that probability. These can be addressed through a closely related algorithm called *max-sum*, which can be viewed as an application of *dynamic programming* in the context of graphical models (Cormen *et al.*, 2001).

A simple approach to finding latent variable values having high probability would be to run the sum-product algorithm to obtain the marginals $p(x_i)$ for every variable, and then, for each marginal in turn, to find the value x_i^* that maximizes that marginal. However, this would give the set of values that are *individually* the most probable. In practice, we typically wish to find the set of values that *jointly* have the largest probability, in other words the vector \mathbf{x}^{\max} that maximizes the joint distribution, so that

$$\mathbf{x}^{\max} = \arg \max_{\mathbf{x}} p(\mathbf{x}) \tag{8.87}$$

for which the corresponding value of the joint probability will be given by

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}). \tag{8.88}$$

In general, \mathbf{x}^{\max} is not the same as the set of x_i^* values, as we can easily show using a simple example. Consider the joint distribution $p(x, y)$ over two binary variables $x, y \in \{0, 1\}$ given in Table 8.1. The joint distribution is maximized by setting $x = 1$ and $y = 0$, corresponding the value 0.4. However, the marginal for $p(x)$, obtained by summing over both values of y , is given by $p(x = 0) = 0.6$ and $p(x = 1) = 0.4$, and similarly the marginal for y is given by $p(y = 0) = 0.7$ and $p(y = 1) = 0.3$, and so the marginals are maximized by $x = 0$ and $y = 0$, which corresponds to a value of 0.3 for the joint distribution. In fact, it is not difficult to construct examples for which the set of individually most probable values has probability zero under the joint distribution.

We therefore seek an efficient algorithm for finding the value of \mathbf{x} that maximizes the joint distribution $p(\mathbf{x})$ and that will allow us to obtain the value of the joint distribution at its maximum. To address the second of these problems, we shall simply write out the max operator in terms of its components

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \dots \max_{x_M} p(\mathbf{x}) \tag{8.89}$$

Exercise 8.27

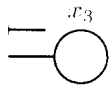


Figure 8.51. (a) nodes.

we can now marginal $p(x_2)$ is messages using

$$x_4)$$

(8.86)

dden. In most we wish to cal- ved nodes are we partition \mathbf{x} we partition \mathbf{x} ed value of \mathbf{v} $\prod_i I(v_i, \hat{v}_i)$, t corresponds $\hat{\mathbf{v}}$). By run- for marginals and efficiently collapse into a

discrete vari- the graphical gorithm. For