Chapter 4

Propagation in Bayesian networks

This chapter presents the algorithm used in HUGIN for probability updating in Bayesian networks. The algorithm does not work directly on the Bayesian network, but on a so-called *junction tree* which is a tree of clusters of variables. The clusters are also called *cliques* because they are cliques in a *triangulated graph*, which is a special graph constructed over the network. Each clique holds a table over the configurations of its variables, and HUGIN propagation consists of a series of operations on these tables. The subjects in this chapter are rather mathematical, and the reader interested in the results rather than in the reasoning behind them can jump directly to the summary in Section 4.7, which should give sufficient background for the reading of Chapters 5 and 6.

In Section 4.1 we define the multiplication and division of tables to be used in the algorithm. Section 4.2 gives methods for entering evidence and updating probabilities provided the full joint probability table is available, and in Section 4.3 we give the architecture of the algorithm when the cluster tree is available. Section 4.4 defines the concept junction tree, and we prove the correctness of the algorithm when applied on a junction tree. Section 4.5 is devoted to the construction of a junction tree from the Bayesian network.

The HUGIN algorithm yields the exact updated probabilities, but if you are unlucky, the algorithm will require so much space or time that the task is intractable. In Section 4.6 we present a technique, *stochastic simulation,* which can be used to get approximate probabilities when this happens.

4.1 An algebra of belief tables

Before we treat probability updating, we will introduce more formally the multiplication of belief tables, which we have used implicitly already.

Table 4.1 Multiplication of two tables over $\{A, B\}$; both variables are ternary.

	a_1	a_2	a_3
b_1	x_1	x_2	x_3
b_2	y_1	y_2	y_3
b_3	z_1	z_2	z_3

t

	a_1	a_2	a_3
b_1	x'_1	x'_2	x'_3
b_2	y'_1	y'_2	y'_3
b_3	z'_1	z'_2	z'_3

t′

	a_1	a_2	a_3
b_1	$x_1x'_1$	$x_2x'_2$	$x_3x'_3$
b_2	$y_1y'_1$	$y_2y'_2$	$y_3y'_3$
b_3	$z_1z'_1$	$z_2z'_2$	$z_3z'_3$

$\mathbf{t} \cdot \mathbf{t}'$

4.1.1 Multiplication and division

Let $\mathbf{t}$ and $\mathbf{t}'$ be two tables over the same variables. Then the product $\mathbf{t} \cdot \mathbf{t}'(c^*) = \mathbf{t}(c^*) \cdot \mathbf{t}'(c^*)$ for all configurations c^*.

Table 4.1 gives an example.

If the two tables are over different sets of variables we can also perform a multiplication.

Let $\mathbf{t}_{AB}$ be a table over $\{A, B\}$, and let $\mathbf{t}_{AC}$ be a table over $\{A, C\}$. Then $\mathbf{t}_{AB}$ and $\mathbf{t}_{AC}$ are multiplied by constructing a table $\mathbf{t}_{ABC}$ over $\{A, B, C\}$, and letting $\mathbf{t}_{AB} \cdot \mathbf{t}_{AC}(a, b, c) = \mathbf{t}_{AB}(a, b) \cdot \mathbf{t}_{AC}(a, c)$ for all configurations (a, b, c).

See Table 4.2 for an example.

Table 4.2 Multiplication of $\mathbf{t}_{AB}$ with $\mathbf{t}_{AC}$.

	a_1	a_2
b_1	x_1	x_2
b_2	x_3	x_4

$\mathbf{t}_{AB}$

	a_1	a_2
c_1	y_1	y_2
c_2	y_3	y_4

$\mathbf{t}_{AC}$

	a_1	a_2
b_1	(x_1y_1, x_1y_3)	(x_2y_2, x_2y_4)
b_2	(x_3y_1, x_3y_3)	(x_4y_2, x_4y_4)

$\mathbf{t}_{AB} \cdot \mathbf{t}_{AC}$

Division can be performed in the same way. Only, we have to be careful with zeros. If the denominator table has zero-entries, then the numerator table must have zero at the same places. In that case we put $\frac{0}{0} = 0$.

4.1.2 Marginalization

Let $\mathbf{t}_V$ be a table over V, and let W be a subset of V. A table $\mathbf{t}_W$ over W can be constructed by *marginalization*. For each configuration w^* let $\mathbf{t}_W(w^*)$ be the sum of all $\mathbf{t}_V(v^*)$, where v^* is a configuration of V coinciding with w^*. The notation is

$$\mathbf{t}_W = \sum_{V \setminus W} \mathbf{t}_V.$$

We shall use the following proposition later.

Proposition 4.1 *Let W and V be disjoint sets of variables, and let $\mathbf{t}_W$ and $\mathbf{t}_V$ be tables over W and V. Then*

$$\sum_V (\mathbf{t}_W \cdot \mathbf{t}_V) = \mathbf{t}_W \cdot \sum_V \mathbf{t}_V.$$

That is, tables containing only variables over which you do not marginalize can be taken out of marginalization. See Table 4.3 for an example.

Table 4.3 An example that $\sum_A \mathbf{t}_B \cdot \mathbf{t}_A = \mathbf{t}_B \sum_A \mathbf{t}_A$.

$$\mathbf{t}_B = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix} \qquad \mathbf{t}_A = \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix}$$

	a_1	a_2	a_3
b_1	y_1x_1	y_1x_2	y_1x_3
b_2	y_2x_1	y_2x_2	y_2x_3
b_3	y_3x_1	y_3x_2	y_3x_3

$\mathbf{t}_B\mathbf{t}_A$

$$\sum_A \mathbf{t}_B\mathbf{t}_A = \begin{array}{c|c|} b_1 & y_1x_1 + y_1x_2 + y_1x_3 \\ b_1 & y_2x_1 + y_2x_2 + y_2x_3 \\ b_1 & y_3x_1 + y_3x_2 + y_3x_3 \end{array} \qquad \mathbf{t}_B \sum_A \mathbf{t}_A = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix} (x_1 + x_2 + x_3)$$

4.2 Probability updating in joint probability tables

Let A be a variable with $P(A) = (x_1, \ldots x_n)$. Assume we get the information e that A can only be in states i and j. This statement says that all states except i and j are impossible, and we have the belief $P(A, e) = (0, \ldots, 0, x_i, 0, \ldots, x_j, 0, \ldots, 0)$. Note that $P(e)$, the prior probability of e, is $x_i + x_j$, the sum of the probabilities of the possible states. To calculate $P(A \mid e)$ we use the fundamental rule:

$$P(A \mid e) = \frac{P(A, e)}{P(e)} = \frac{P(A, e)}{\sum_A P(A, e)}.$$

The way that e is entered can be interpreted as a multiplication of $P(A)$ with the table $\underline{e} = (0, \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0)$ resulting in $P(A, e)$.

Definition. Let A be a variable with n states. A *finding* on A is an n-dimensional table of zeros and ones.

Semantically, a finding is a statement that certain states of A are impossible.

Now, let U be a universe of variables, and assume that we have easy access to $P(U)$, the joint probability table. Then, $P(B)$ for any variable B in U is easy to calculate:

$$P(B) = \sum_{U \setminus \{B\}} P(U).$$

Suppose we wish to enter the above finding. Then $P(U, e)$ is the table resulting from $P(U)$ by giving all entries with A in state i or j the value zero and leaving the other entries unchanged. Again, $P(e)$ is the sum of all entries in $P(U, e)$ and

$$P(U \mid e) = \frac{P(U, e)}{P(e)} = \frac{P(U, e)}{\sum_U P(U, e)}.$$

Note that $P(U, e)$ is the product of $P(U)$ and the finding $\underline{e}$. If e consists of several findings $\{f_1, \ldots, f_m\}$ each finding can be entered separately, and $P(U, e)$ is the product of $P(U)$ and the findings $\underline{f}_i$. We can express the considerations above in the following theorem.

Theorem 4.1 *Let U be a universe of variables and let $e = \{f_1, \ldots, f_m\}$. Then*

$$P(U, e) = P(U) \cdot \underline{f}_1 \cdot \cdots \cdot \underline{f}_m \text{ and } P(U, e) = \frac{P(U \mid e)}{P(e)},$$

where

$$P(e) = \sum_U P(U, e).$$

Theorem 4.1 says that if we have access to $P(U)$, then we can enter evidence and perform probability updating. However, even for small sets of variables, the table $P(U)$ is intractably large, and we have to find a smaller representation.

4.3 Cluster trees

As shown in Section 2.3.7 (the chain rule), a Bayesian network over U is a representation of $P(U)$. This means that we can, in principle, calculate $P(U)$ as the product of all conditional probabilities from the network. The question then, is whether we can enter evidence and perform probability updating in Bayesian networks without being forced to calculate $P(U)$. It has turned out to be rather difficult.

Instead we can work with another representation called *cluster trees*.

Definition. A *cluster tree over* U is a tree of clusters of variables from U. The nodes are subsets of U, and the union of all nodes is U. (A tree is an undirected graph without cycles.)

The links are labelled with *separators* which consist of the intersection of the adjacent nodes.

Each node and separator holds a real numbered table over the configurations of its variable set.

In Figure 4.1 we give a cluster tree for the network $M_{\min}$

Figure 4.1 The Bayesian network $M_{\min}$ and a corresponding cluster tree. Separators are in square boxes.

Now, let *BN* be a Bayesian network over U. A *Cluster tree corresponding to BN* is constructed in the following way:

- form a family of nodes such that for each variable A with parent set $pa(A)$ there is at least one node V such that $pa(A) \cup \{A\} \subseteq V$;
- organize the nodes as a tree with separators (so far there is no restriction on how you organize the tree);
- give all nodes and separators a table of ones.
- for each variable A choose exactly one node V containing $pa(A) \cup \{A\}$ and multiply $P(A \mid pa(A))$ on Vs table.

Then the product of all node tables in the cluster tree is the product of all conditional probability tables in *BN*, and therefore we have the following theorem.

Theorem 4.2 *Let* BN *be a Bayesian network over U. Then any cluster tree corresponding to* BN *is a representation of $P(U)$, and $P(U)$ is the product of all cluster tables divided by the product of all separator tables.*

Remark. In Theorem 4.2 we divide the product of all cluster tables by the product of all separator tables. This does not do any harm, because the separator tables consist of ones, but the reader may wonder why. The reason is that, when we now start to move the information around in the cluster tree, then the product of all cluster tables divided by all separator tables is invariant, and thereby the tree remains a representation of $P(U)$.

It is easy to insert findings into a cluster tree. Let e be a finding on A. Multiply $\underline{e}$ on the table of any node containing A. Then, by the chain rule and Theorem 4.1 the product of all node tables is $P(U) \cdot \underline{e} = P(U, e)$.

To calculate $P(B, e)$ for an arbitrary variable B is not as easy, and the coming sections are devoted to this problem.

4.3.1 Absorption in cluster trees

We introduce an operation in cluster trees. It has the effect of re-arranging the information stored in the tables.

Definition. Let V and W be neighbours in a cluster tree, let S be their separator, and let $\mathbf{t}_V$, $\mathbf{t}_W$ and $\mathbf{t}_S$ be their tables. The operation *absorption* is the result of the following procedure:

- calculate $\mathbf{t}_S^* = \sum_{V \setminus S} \mathbf{t}_V$;
- give S the table $\mathbf{t}_S^*$;
- give W the table $\mathbf{t}_W^* = \mathbf{t}_W \frac{\mathbf{t}_S^*}{\mathbf{t}_S}$.

We then say that W has *absorbed* from V or that W calibrates to V.

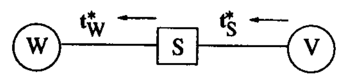

Figure 4.2 W absorps from V. $\mathbf{t}^*_S = \sum_{V\setminus S} \mathbf{t}_V$; $\mathbf{t}^*_W = \mathbf{t}_W \cdot \frac{\mathbf{t}^*_S}{\mathbf{t}_S}$.

Remarks.

(1) The idea behind absorption is that the information which V and W can ha in common is the information on S, and this is what W receives from V. W, V and S hold the same information on S, that is if

$$\sum_{W\setminus S} \mathbf{t}_W = \mathbf{t}_S = \sum_{V\setminus S} \mathbf{t}_V,$$

then absorption does not change anything. We then say that the link is *co sistent.* If all links in the cluster tree are consistent we say that the tree consistent. If a tree is consistent, then absorption does not have any effect all.

Assume that the link is consistent, but now some evidence changes $\mathbf{t}_V$ $\mathbf{t}^*_V$. Then after W has absorbed from V, the three tables all hold Vs ne information on S:

$$\sum_{W\setminus S} \mathbf{t}^*_W = \sum_{W\setminus S} \mathbf{t}_W \frac{\mathbf{t}^*_S}{\mathbf{t}_S} = \frac{\mathbf{t}^*_S}{\mathbf{t}_S} \sum_{W\setminus S} \mathbf{t}_W = \frac{\mathbf{t}^*_S}{\mathbf{t}_S} \mathbf{t}_S = \mathbf{t}^*_S = \sum_{V\setminus S} \mathbf{t}^*_V.$$

(2) W can only absorb from V through S if $\mathbf{t}_W$ has zeros in the entries correspo ing to the zero-entries in $\mathbf{t}_S$. We say that a link in a cluster tree is *support* if it allows absorption in both directions, and a cluster tree is supportive if a its links are supportive. Note that the cluster trees constructed in Section 4. are supportive since the separator tables have no zero-entries.

Lemma 4.1 *Supportiveness is preserved under absorption.*

Proof. Let W absorb from V through the separator S. Then

$$\mathbf{t}^*_W = \mathbf{t}_W \cdot \frac{\mathbf{t}^*_S}{\mathbf{t}_S},$$

where

$$\mathbf{t}^*_S = \sum_{V\setminus S} \mathbf{t}_V.$$

Then any zero-entry in $\mathbf{t}^*_S$ is also a zero-entry in $\mathbf{t}^*_W$. This clearly also holds for t

Theorem 4.3 *Let T be a supportive cluster tree. Then the product of all clus tables divided by the product of all separator tables is invariant under absorption.*

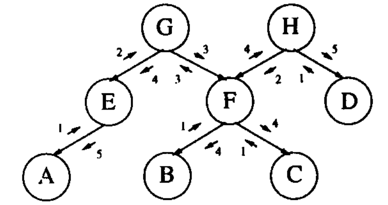

Figure 4.3 Certainty updating through message passing in a cluster tree. The numbers on the links indicate the order in which the messages are passed and in which direction.

Proof. When W absorbs from V through the separator S, only the tables of W and S are changed. Therefore it is enough to prove that the fraction of Ws and Ss table is unchanged. We have

$$\frac{\mathbf{t}^*_W}{\mathbf{t}^*_S} = \frac{\mathbf{t}_W \cdot \frac{\mathbf{t}^*_S}{\mathbf{t}_S}}{\mathbf{t}^*_S} = \frac{\mathbf{t}_W}{\mathbf{t}_S}.$$

Theorem 4.3 ensures that if we start with a Bayesian network over U, construct a corresponding cluster tree T, and then perform a series of absorptions, then T remains a representation of $P(U)$, and $P(U)$ can be calculated as the product of all cluster tables divided by the product of all separator tables.

4.3.2 Message passing in cluster trees

The next question is how many absorptions can we perform, and can they help us in transforming the tables in a cluster tree into a form where it is easy to calculate $P(A)$ for single variables?

We can think of absorptions as messages passed between the nodes in the tree. That is, a node V *sends a message* to its neighbour W when W absorbs from V.

Message passing scheme. A node V can send exactly one message to a neighbour W, and it may only be sent when V has received a message from each of its other neighbours.

Consider, for example, the cluster tree in Figure 4.3. The leaves of the tree (the nodes A, B, C, D) can send to their single neighbour (1). Then E can send to G, and H can send to F (2). Next, G can send to F, and F can send to G (3), F can send to H, B and C, and G can send to E (4). Finally E can send to A and H to D (5). Now each node has sent to all of its neighbours.

As can be seen, the message passing algorithm is not sequential, and a good way of thinking of it is that each variable is busy waiting, eager to send messages. Each time it receives a message it updates its own table and sends a message to the eligible neighbours (if any).

Theorem 4.4 *Let T be a supportive cluster tree, and suppose that messages are passed according to the message passing scheme. Then:*

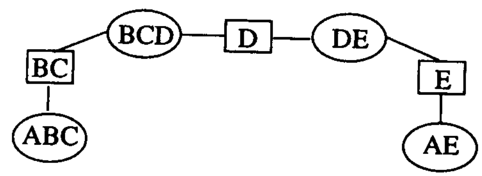

Figure 4.4 A cluster tree over binary variables. All variables except A are in state y. In the node (A, B, C) A is in state y, and in the node (A, E) A is in state n. Though the cluster tree is consistent, the table for $\mathbf{t}_A$ marginalized from $\mathbf{t}_{ABC}$ is different from the marginal taken from $\mathbf{t}_{AE}$.

(i) message passing can continue until a message has been passed in both dir- tions of each link;

(ii) when a message has been passed in both directions of each link then T consistent.

Proof. (i) Exercise 4.3.
(ii) If T consists of only one node then the theorem is obviously true.

Assume that T has more than one node, and let (V, W) be an arbitrary link wi separator S. Let the first message to be passed over (V, W) be from W to V, an let $\mathbf{t}_V$, $\mathbf{t}_S$ and $\mathbf{t}_W$ be the tables before the message is passed.
When the message has been passed we have $\mathbf{t}_S^* = \sum_{W\setminus S} \mathbf{t}_W$. Next, when th message from V and W has to be passed, the tables for S and W have not be changed (W has not received further messages). Let the table for V be $\mathbf{t}_V^*$. After message passing we have

$$\mathbf{t}_S^{**} = \sum_{V\setminus S} \mathbf{t}^{**} \quad \textit{and} \quad \mathbf{t}_W^{**} = \mathbf{t}_W \frac{\mathbf{t}_S^{**}}{\mathbf{t}_S^*}.$$

Now

$$\sum_{V\setminus S} \mathbf{t}_V^* = \sum_{V\setminus S} \mathbf{t}_V \frac{\mathbf{t}_S^{**}}{\mathbf{t}_S^*} = \frac{\mathbf{t}_S^{**}}{\mathbf{t}_S^*} \sum_{V\setminus S} \mathbf{t}_V = \frac{\mathbf{t}_S^{**}}{\mathbf{t}_S^*} \mathbf{t}_S^* = \mathbf{t}_S^{**} = \sum_{W\setminus S} \mathbf{t}_W^{**}.$$

Therefore the link is consistent.

4.4 Junction trees

Let T be a cluster tree over U, let A be a variable in U, and suppose that A is element of the nodes V and W. If T is consistent we would expect $\sum_{V\setminus\{A\}} \mathbf{t}_V$ $\sum_{W\setminus\{A\}} \mathbf{t}_W$. Certainly this is so if V and W are neighbours, but otherwise there no guarantee. See Figure 4.4 for an example.
We say that a consistent cluster tree is *globally consistent* if for any nodes V W with intersection I we have

$$\sum_{V\setminus I} \mathbf{t}_V = \sum_{W\setminus I} \mathbf{t}_W.$$

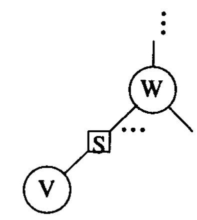

Figure 4.5 V is a leaf of T linked to W and with separator S.

As Figure 4.4 indicates, the reason why consistence does not imply global consistence is that a variable A can be placed in two locations in the tree such that information on A cannot be passed between the two locations. To ensure global consistence we must add a requirement to cluster trees.

Definition. A cluster tree is a *junction tree* if, for each pair of nodes V, W, all nodes on the path between V and W contain the intersection $V \cap W$.

Theorem 4.5 *A consistent junction tree is globally consistent.*

Proof. Exercise 4.7. □

The following theorems will show that if we construct a junction tree corresponding to a Bayesian network, then we have good algorithms for insertion of evidence as well as probability updating. When we construct a cluster tree corresponding to a Bayesian network we have several degrees of freedom, and we shall use them for constructing a junction tree. However, it is not easy. For example, with the clusters in Figure 4.4 it is impossible to construct a tree with the junction tree property. We will leave this problem here, and return to it in Section 4.5.

Theorem 4.6 *Let T be a consistent junction tree over U, and let $\mathbf{t}_U$ be the product of all node tables divided by the product of all separator tables. Let V be a node with table $\mathbf{t}_V$. Then*

$$\mathbf{t}_V = \sum_{U\setminus V} \mathbf{t}_U.$$

Proof. Induction on the number of nodes.
Clearly the theorem holds when T consists of a single node.
Now, assume the theorem to hold for any junction tree with n nodes, and let T be a consistent junction tree with $n + 1$ nodes. Let V be a leaf of T linked to W and with separator S (see Fig. 4.5). Let T' be the junction tree resulting from removing V (and S), and let T' have the universe U'. Then

$$\mathbf{t}_U = \mathbf{t}_{U'} \cdot \frac{\mathbf{t}_V}{\mathbf{t}_S}.$$

where $\mathbf{t}_{U'}$ is the product of all node tables in T' divided by the separator tables in T'. Let D be the set of variables $V \setminus S$, and let H be $W \setminus S$. From the junction tree property we have that $D \cap U' = \emptyset$.

Since T is consistent we have

$$\sum_D \mathbf{t}_V = \mathbf{t}_S = \sum_H \mathbf{t}_W.$$

Now

$$\begin{aligned}\sum_D \mathbf{t}_U &= \sum_D \mathbf{t}_{U'} \cdot \frac{\mathbf{t}_V}{\mathbf{t}_S} \\ &= \mathbf{t}_{U'} \cdot \frac{\sum_D \mathbf{t}_V}{\mathbf{t}_S} \\ &= \mathbf{t}_{U'} \cdot \frac{\mathbf{t}_S}{\mathbf{t}_S} \\ &= \mathbf{t}_{U'}.\end{aligned}$$

Therefore, by the induction hypothesis we have

$$\sum_{U \setminus V_i} \mathbf{t}_U = \mathbf{t}_{V_i}$$

for all V_i in T'.

Furthermore,

$$\begin{aligned}\sum_{U \setminus V} \mathbf{t}_U &= \sum_{U' \setminus S} \mathbf{t}_{U'} \cdot \frac{\mathbf{t}_V}{\mathbf{t}_S} \\ &= \frac{\mathbf{t}_V}{\mathbf{t}_S} \cdot \sum_{U' \setminus S} \mathbf{t}_{U'} \\ &= \frac{\mathbf{t}_V}{\mathbf{t}_S} \cdot \sum_{W \setminus S} \mathbf{t}_W \\ &= \frac{\mathbf{t}_V}{\mathbf{t}_S} \cdot \mathbf{t}_S \\ &= \mathbf{t}_V.\end{aligned}$$

The considerations above are summarized in the following theorem.

Theorem 4.7 *Let* BN *be a Bayesian network representing $P(U)$, and let T b[e a] junction tree corresponding to* BN. *After a full round of message passing in T, [we] have for each node V and each separator S that*

$$\mathbf{t}_V = \sum_{U \setminus V} P(U) = P(V) \text{ and } \mathbf{t}_S = P(S).$$

Proof. By Theorem 4.2, $P(U)$ is the product of the initial node tables divided [by] the separator tables. Theorems 4.3 and 4.4 give that after a full round of mess[age] passing T is consistent, and $P(U)$ is the product of all node tables divided by [the] separator tables. Theorems 4.5 and 4.6 yield the conclusion.

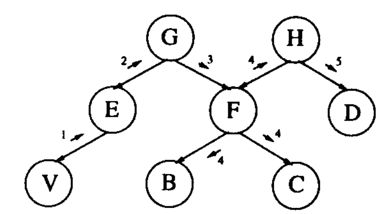

Figure 4.6 The message passing in *DistributeEvidence*(V).

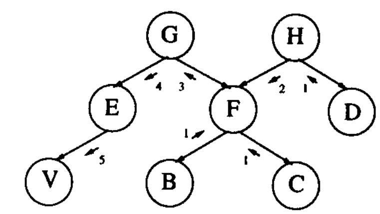

Figure 4.7 The message passing in *CollectEvidence*(V).

Theorem 4.8 *Let* BN *be a Bayesian network representing $P(U)$, and let T be a junction tree corresponding to* BN. *Let $e = \{f_1, \ldots, f_m\}$ be findings on the variables $\{A_1, \ldots, A_m\}$. For each i find a node containing A_i and multiply its table with $\underline{f}_i$.*

Then, after a full round of message passing we have for each node V and separator S that

$$\mathbf{t}_V = P(V, e) \qquad \mathbf{t}_S = P(S, e) \qquad P(e) = \sum_V \mathbf{t}_V.$$

Proof. Use Theorem 4.1 and proceed as in the proof of Theorem 4.7. □

4.4.1 HUGIN propagation

Assume that we have a consistent junction tree, and now a single node V receives evidence. Then half of the messages can be avoided: V sends messages to all of its neighbours who recursively send messages to all neighbours except the one from which the message came (see Fig. 4.6). We call this algorithm *DistributeEvidence*.

Now, suppose that we are only interested in the certainty of one node, V. Then half of the certainty updating messages can be avoided: V asks all its neighbours to send it a message, and if they are not allowed to do so, they recursively pass the request to all neighbours except the one from which the request came (see Fig. 4.7). We call this algorithm *CollectEvidence*.

The two algorithms *DistributeEvidence* and *CollectEvidence* can be used for a more organized message passing scheme. No matter the amount of evidence entered, take any variable V. Call *CollectEvidence* from V and after that call *DistributeEvidence* from V. The result is that all messages have been passed, and they were passed when permitted (see Fig. 4.8 and Exercise 4.5).

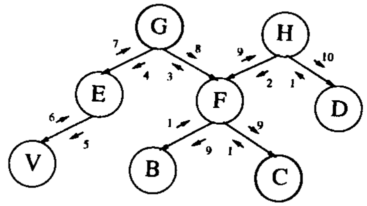

Figure 4.8 Updating through *CollectEvidence*(V) followed by *DistributeEvidence*(V).

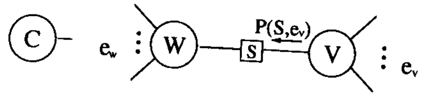

Figure 4.9 Evidence e_V has been entered at the righthand side of S. e_W has been entered at the lefthand side of S. C is used as a root for the propagation.

HUGIN propagation uses corresponding junction trees, and the operations *Collect-Evidence* and *DistributeEvidence*. A node Rt in the junction tree is chosen as a root, and whenever a propagation takes place, *CollectEvidence*(Rt) is called followed by a call of *DistributeEvidence*(Rt). When the calls are finished, the tables are *normalized* so that they sum to one.

HUGIN propagation has a nice side effect, namely that it gives access to various probabilities of sets of entered findings.

Let us use Theorem 4.8 to have a closer look at what is actually communicated in the propagation algorithm. The general situation is described in Figure 4.9.

A call of *CollectEvidence*(C) will cause a call of *CollectEvidence*(V), and by Theorem 4.8 this will result in $\mathbf{t}^*_V = P(V, e_V)$. This gives that $P(e_V)$ can be calculated without further propagations. Unfortunately, the situation is not symmetric. In the *DistributeEvidence* phase the message passed from W to S is $P(S, e)$.

4.5 Construction of junction trees

In this section we shall give a method for constructing junction trees for DAGs.

4.5.1 Singly connected DAGs

A DAG is *singly connected* if the graph you get by dropping the directions of the links is a tree (see Fig. 4.10).

For singly connected DAGs it is easy to construct junction trees. For each varia[ble] A with $pa(A) \neq \emptyset$ you form the cluster $pa(A) \cup \{A\}$. Between any two clus[ters] with a non-empty intersection you add a link with the intersection as a separ[ator]. The resulting graph is called a *junction graph*. All separators consist of a si[ngle] variable, and if the junction graph has cycles, then all separators on the cycle con[sist of]

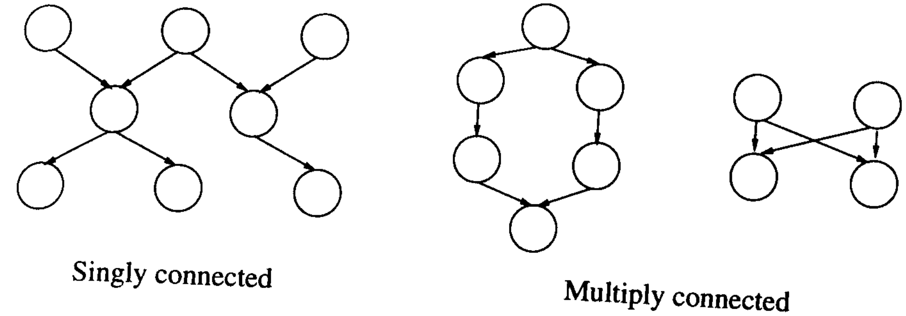

Figure 4.10 Examples of singly connected and multiply connected DAGs.

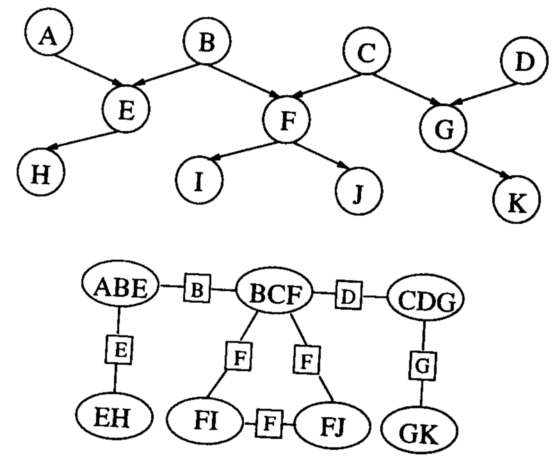

Figure 4.11 A singly connected DAG and its junction graph. By removing any of the links with separator F you get a junction tree.

the same variable. Therefore any of the links can be removed to break the cycle, and by removing links until you have a tree, you get a junction tree (see Fig. 4.11).

We know that when we construct a cluster tree corresponding to a DAG, then for all variables A there must be a cluster V containing $pa(A) \cup \{A\}$. We can illustrate this on a graph by having a link between any pair of variables which must appear in the same cluster. This means that we take the DAG, add a link between any pair of variables with a common child, and drop the directions of the original links. The resulting graph is called the *moral graph*. From the moral graph you can read the clusters to consider, namely the cliques in the graph (maximal sets of variables that are all pairwise linked). In Figure 4.12 we give an example of the construction.

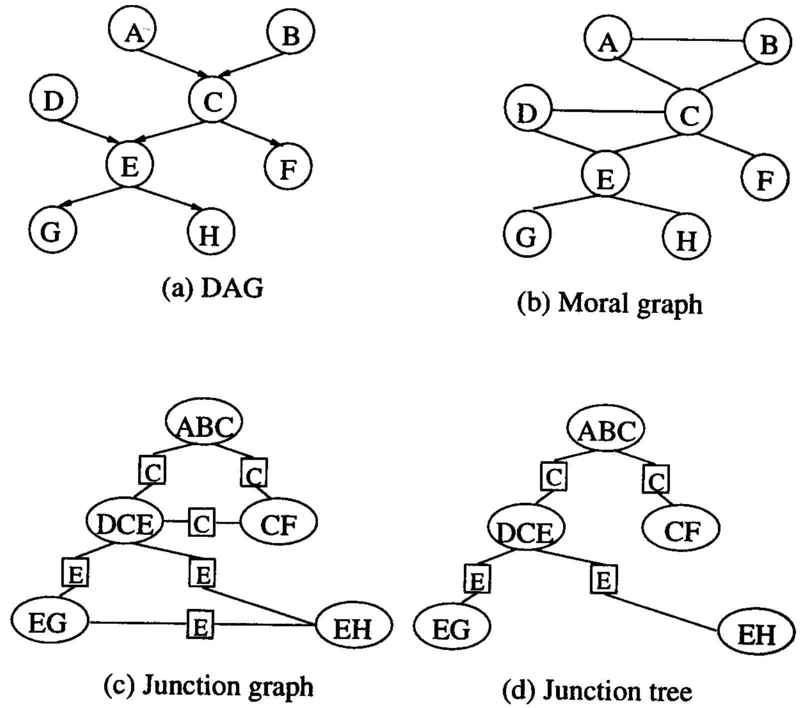

Figure 4.12 Construction of a junction tree for a singly connected DAG.

4.5.2 Coping with cycles

Consider the junction graph in Figure 4.13. The intersection of the two clusters variables is (AB), and a junction tree is easily found.

Consider the DAG in Figure 4.14(a) with the moral graph in Figure 4.14(Sticking to the approach that the clusters are the cliques in the moral graph, we that if we join A, B and C, then we get a junction tree.

The DAG in Figure 4.15 is more problematic. The cycle in the junction gr cannot be broken.

The propagation problem is that coupled information (on (DE)) is decoupled meets again under propagation. This can also be seen from the cycle $D - E - C$ $A - B - D$ in the moral graph. A way to solve the problem is to add so-ca *fill-ins* to the moral graph: add a link between C and D and one between B and The result is shown in Figure 4.16 together with the resulting junction tree.

The general rule for filling-in the moral graph is that any cycle with more three variables shall have a chord. In this case the graph is called *triangulated*.

In Figures 4.17 and 4.18 there is another example of the process from DAG junction tree. Note that without the fill-in $(B - D)$ the cycle $A - B - F - D$ does not have a chord.

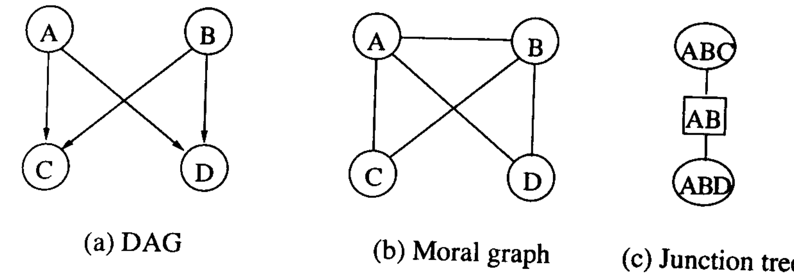

Figure 4.13 Construction of a junction tree for a simple multiply connected DAG.

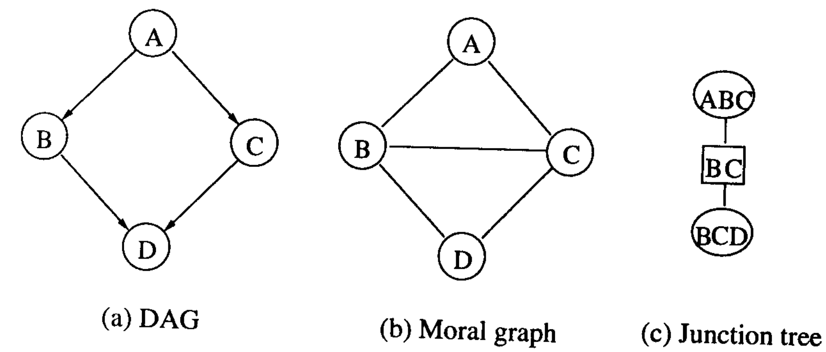

Figure 4.14 Another simple DAG with a cycle.

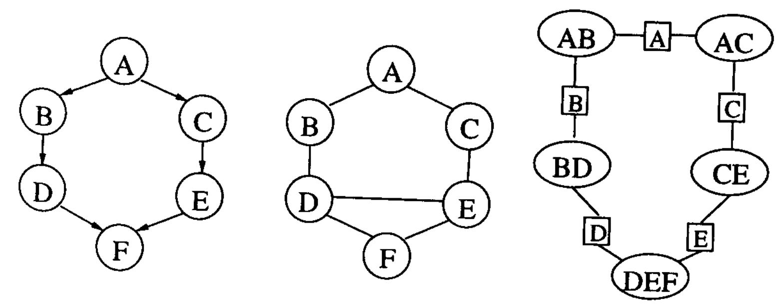

Figure 4.15 A DAG with a large cycle.

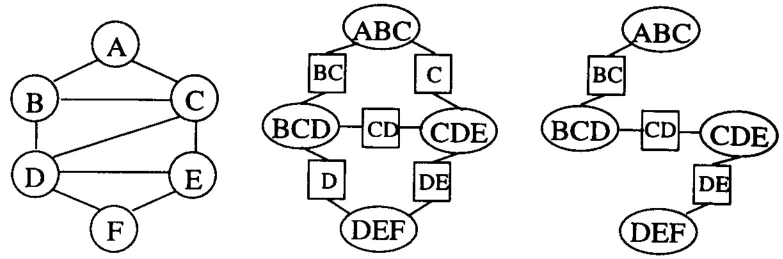

Figure 4.16 The filled-in moral graph from Figure 4.15, the junction graph, and the juction tree resulting from removing the links with separator D and C.

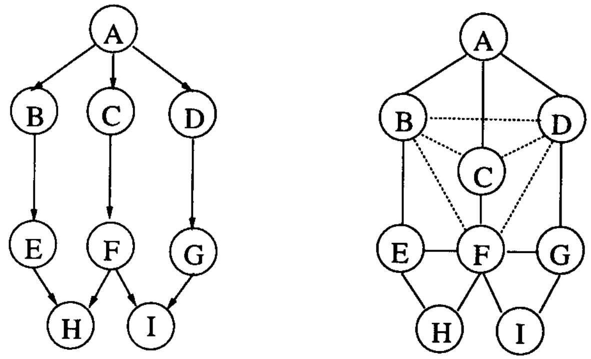

Figure 4.17 A DAG, the moral and triangulated graphs. The fill-ins are indicated by dotted lines.

4.5.3 From DAG to junction tree

In this section we present, without proofs, algorithms for triangulation of graphs an
for construction of junction trees from triangulated graphs. Proofs of Theorems 4
and 4.10 are given in Appendix A.

Definition. An undirected graph is *triangulated* if any cycle of length > 3 has
chord.

Definition. A node A is *eliminated* by adding links such that all of its neighbo
are pairwise linked and then removing A together with its links.

Note that if a node A can be eliminated without adding links, then A cannot
part of a chordless cycle of length > 3.

Theorem 4.9 *A graph is triangulated if and only if all of its nodes can be elimina
one by one without adding any link.*

Theorem 4.9 yields a method for triangulation as well as a test for whether a gr
is triangulated. The method consists of eliminating the nodes in some order (add

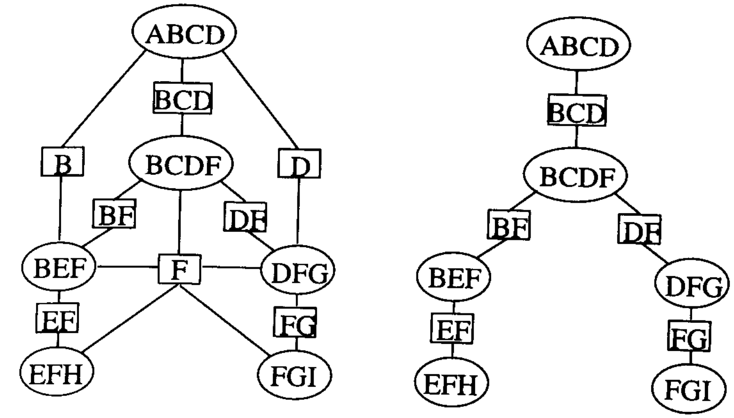

Figure 4.18 The junction graph for the triangulated graph in Figure 4.17 and a junction tree.

links, if necessary) and when this is done the resulting graph is triangulated. An example is given in Figure 4.20.

Note that there are several triangulations of the graph. Intuitively, triangulations with as few fill-ins as possible are preferred. However, optimality is connected to the resulting junction tree and the computational complexity of the propagation algorithm. We shall return to the question of optimality later.

Definition. A *junction graph* for an undirected graph G is an undirected, labelled graph. The nodes are the cliques in G. Every pair of nodes with a non-empty intersection has a link labelled by the intersection.

There is an easy way of identifying the cliques in a triangulated graph G. Let $A_1, \ldots, A_n$ be an elimination sequence for G, and let C_i be the set of variables containing A_i and all its neighbours at the time of elimination (neighbours with higher numbers). Then every clique of G is a C_i for some i.

The reader may check that the cliques of the graphs in Figure 4.20(a) are C_1, C_2, C_3, C_4, and that the cliques of the graph in Figure 4.20(b) are C_1, C_2, C_3.

The junction tree we are aiming at will be a subgraph of the junction graph. Since message passing will be restricted to links in the junction tree we are not allowed to remove a link from the junction graph if thereby some kind of information cannot be passed. If, for example, the clusters U and V have the variable A in common, they have a link with label A. If this link is removed, there shall be another path in the remaining graph through which information on A can be passed from U to V. So, let us recall the following definition.

Definition. A spanning tree of a junction graph is a *junction tree* if it has the property that for each pair of nodes, U, V, all nodes on the path between U and V contain $U \cap V$. (A subtree of a graph is a spanning tree if all nodes of the graph are nodes in it.)

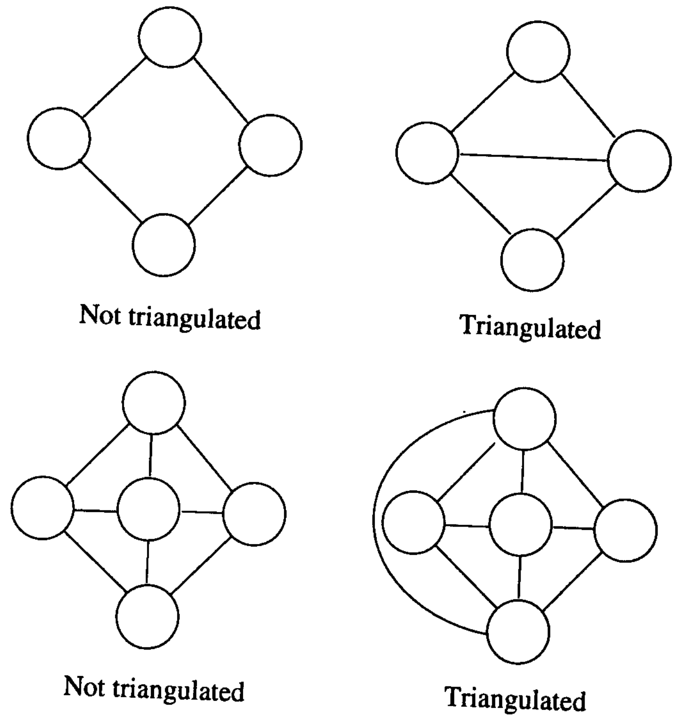

Figure 4.19 Triangulated and not triangulated graphs.

Theorem 4.10 *An undirected graph is triangulated if and only if its junction graph has a junction tree.*

Definition. The *weight* of a link in a junction graph is the number of variables in the label. The weight of a junction tree is the sum of the weights of the labels.

Theorem 4.11 *(Without proof.) A subtree of the junction graph of a triangulated graph is a junction tree if and only if it is a spanning tree of maximal weight.*

Theorem 4.11 provides an easy way of constructing junction trees, namely Kruskal's algorithm: choose successively a link of maximal weight unless it creates a cycle.

There are other ways of constructing junction trees. In particular, if an elimination sequence for the triangulated graph is known, very efficient algorithms exist (Exercise 4.8). So, if the graph is triangulated then the construction of a junction tree is rather fast.

The only problematic step in the process from DAG to junction tree is the triangulation. Since any elimination sequence will produce a triangulation it may seem a problem, but for the propagation algorithm it is. In HUGIN propagation, cliques in the junction graph shall have joint probability tables attached to them. The size of the table is the product of the number of states of the variables. So the size increases exponentially with the size of the clique. A good triangulation,

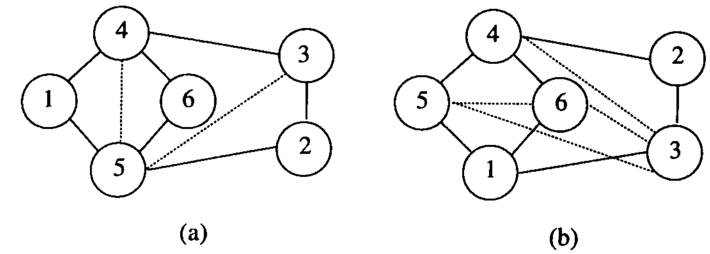

Figure 4.20 Two examples of triangulation through elimination. The numbers on the nodes indicate the elimination order, and the dotted lines are fill-ins.

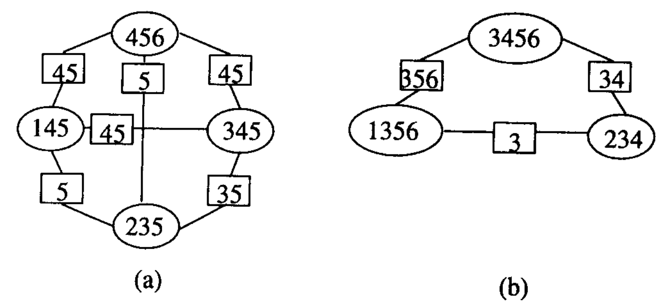

Figure 4.21 Junction graphs for the two triangulated graphs in Figure 4.20.

therefore, is a triangulation yielding small cliques, or to be more precise, yielding small probability tables. The problem of determining an optimal triangulation is NP-complete. However, there is a heuristic algorithm which has proven to give fairly good results. It is a version of the greedy approach: eliminate repeatedly a node not requiring fill-ins and if this is not possible, eliminate a node yielding the smallest table. In Figure 4.23 an example is given.

4.6 Stochastic simulation

The propagation method requires tables for the cliques in the triangulated graph. These cliques may be very large, and it happens that the space requirements cannot be met by the hardware available. In this case an approximate method would be satisfactory.

In this section we shall give a flavour of an approximate method called *stochastic simulation*. The idea behind the simulation is that the causal model is used to simulate the flow of impact. When impact from a set of variables to a variable A is simulated, a random generator is used to decide the state of A.

To illustrate the technique, consider the Bayesian network in Figure 4.24 with the conditional probabilities specified in Table 4.4.

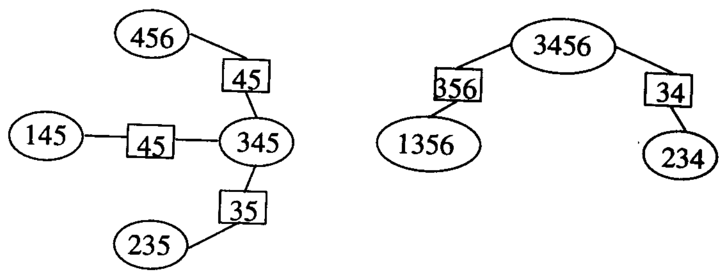

Figure 4.22 Junction trees for the junction graphs in Figure 4.20.

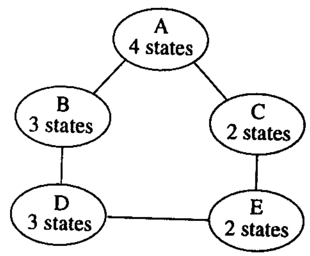

Figure 4.23 A heuristic elimination sequence is E, D (and A, B, C).

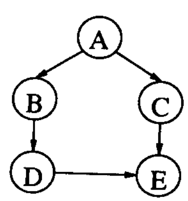

Figure 4.24 An example network. All variables have the states y and n.

Table 4.4 The conditional probabilities for the example network. $P(A) = (0.4, 0.6)$.

	A	
B	y	n
y	0.3	0.8
n	0.7	0.2
$P(B \mid A)$		

	A	
C	y	n
y	0.7	0.4
n	0.3	0.6
$P(C \mid A)$		

	B	
D	y	n
y	0.5	0.1
n	0.5	0.9
$P(D \mid B)$		

	C	
D	y	n
y	(0.9, 0.1)	(0.999, 0.001)
n	(0.999, 0.001)	(0.999, 0.001)
$P(E \mid C, D)$		

Table 4.5 A set of 100 configurations of (A, B, C, D, E) sampled from the network in Figure 4.24 and Table 4.4

	CDE							
AB	yyy	yyn	yny	ynn	nyy	nyn	nny	nnn
yy	4	0	5	0	1	0	2	0
yn	2	0	16	0	1	0	8	0
ny	9	1	10	0	14	0	16	0
nn	0	0	4	0	0	0	7	0

The idea now is to draw a random configuration of the variables (A, B, C, D, E), and to do this a sufficient number of times.

A random configuration is selected by successively sampling the states of the variables. First the state of A is sampled. A random generator (with even distribution) is asked to give a real number between zero and one. If the number is less than 0.4 the state is y, if not the state is n. Assume that the result is y. From the conditional probability table $P(B \mid A)$ we have that $P(B \mid y) = (0.3, 0.7)$. The random generator is asked again, and if the number is less than 0.3, the state of B is y. This procedure is repeated to get the state of C, D, and E, and a configuration is determined.

The next configuration is sampled through the same procedure, and the procedure is repeated until m configurations are sampled. In Table 4.5 an example set of configurations is given.

The probability distributions for the variables are calculated by counting in the sample set (see Exercise 4.12). For 39 of the samples in Table 4.5 the first state is y, and this gives an estimated probability $P(A) = (0.39, 0.61)$.

The method above, called *forward sampling*, does not require a triangulation of the network, and it is not necessary to store the sampled configurations (like Table 4.5); it is enough to store the counts for each variable. Whenever a sampled configuration has been determined, the counts of all variables are updated, and the sample can be discarded. This method saves a great deal of space, and each configuratio determined in a time linear to the number of variables. The cost is accuracy time.

So far only the initial probabilities are calculated. When evidence arrives, it be handled by simply discarding the configurations which do not conform t That is, a new series of stochastic simulations are started, and whenever a sta an observed variable is drawn, you stop simulating if the state drawn is no observed one.

Unfortunately, this method has a serious drawback. Assume in the example a that the observations for the network are $B = n$ and $E = n$. The probability $(B = n, E = n)$ is 0.00282. This means that in order to get 100 configurat you should for this tiny example, expect to perform more than 35 000 stoch simulations.

Methods have been constructed for dealing with this problem. A promising me is called *Gibbs sampling*.

In Gibbs sampling you start with some configuration consistent with the evid (for example determined by forward sampling), and then you randomly chang state of the variables in causal order. In one sweep through the variables determine a new configuration, and then you use this configuration for a new sw etc.

In the example let $B = n$ and $E = n$ be the evidence, and let the sta configuration be *ynyyn*. Now, calculate the probability of A given the other s of that configuration. That is, $P(A \mid B = n, C = y, D = y, E = n)$. From network we see that it is sufficient to calculate $P(A \mid B = n, C = y)$. It is e done by Bayes' rule: it is (0.8, 0.2). We draw a number from the random gener and let us assume that the number is 0.456 resulting in $A = y$. The next free vari is C. We calculate

$$\begin{aligned} P(C \mid A = y, B = n, D = y, E = n) &= P(C \mid A = y, D = y, E = n) \\ &= (0.996, 0.04). \end{aligned}$$

We draw from the random generator, and assume we keep $C = y$.

In general the calculation goes as follows. Let A be a variable in a Bay network BN, let $B_1, \ldots, B_n$ be the remaining variables, and let $b^* = (b_1, \ldots$ be a configuration of $(B_1, \ldots, B_n)$. Then $P(A, b^*)$ is the product of all condi tables of BN with B_i instantiated to b_i. Therefore $P(A, b^*)$ is proportional product of the tables involving A, and $P(A \mid b^*)$ is the result of normalizin product. Note that the calculation of $P(A \mid b^*)$ is a local task.

Back to the example. The next variable is D. We follow the same procedu assume that the result is $D = y$. Then the configuration from the first sw unaltered, i.e. *ynyyn*.

The next sweep follows the same procedure. Assume the result for A is th state is changed to n. Then we shall calculate $P(C \mid A = n, D = y, E = n$ so forth.

In this way a large sample of configurations consistent with the observatio produced. The question is whether the sample is representative for the prob

distribution. It is not always so. It may be that the initial configuration is rather improbable, and therefore the first samples, likewise, are out of the mainstream. Therefore you usually discard the first 5–10% of the samples. It is called *burn-in*.

Another problem is that you may be stuck in certain "areas" of the configurations. Perhaps there is a set of very likely configurations, but in order to reach them from the one you are in, a variable should change to a state which is highly improbable given the remaining configuration (see Exercise 4.13).

A third serious problem is that it may be very hard to find a starting configuration. In fact, it is *NP*-hard (see Exercise 4.14).

We shall not deal with these problems, but refer the interested reader to the literature.

4.7 Summary of Sections 4.2–4.5

Junction trees

The nodes of a junction tree are sets of variables, they are called *cliques*. Each link is labelled with a *separator* which is the intersection of the adjacent cliques. Each clique and separator holds a real numbered table over the configurations of its variable set.

The junction tree property. For each pair V, W of cliques, all cliques on the path between V and W contain the intersection $V \cap W$.

A junction tree is said to *represent* the Bayesian network BN over the variables U if:

(i) for each variable A, there is a clique containing $pa(A) \cup \{A\}$;

(ii) $P(U)$ is the product of all clique tables divided by all separator tables.

Construction of junction trees

Let BN be a Bayesian network over the variables U.

(i) Construct the *moral graph:* the undirected graph with a link between all variables in $pa(A) \cup \{A\}$ for all A.

(ii) *Triangulate* the moral graph: add links until all cycles consisting of more than three links have a chord.

(iii) The nodes of the junction tree are the cliques of the triangulated graph.

(iv) Connect the cliques of the triangulated graph with links such that a junction tree is constructed.

(v) First give all cliques and separators a table consisting of only ones. Then, for each variable A find a clique containing $pa(A) \cup \{A\}$, and multiply $P(A \mid pa(A))$ on its table.

The resulting junction tree represents BN.

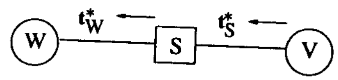

Figure 4.25 W absorps from V. $\mathbf{t}_W^* = \mathbf{t}_W \cdot \frac{\mathbf{t}_S^*}{\mathbf{t}_S}$, $\mathbf{t}_S^* = \sum_{V \setminus S} \mathbf{t}_V$.

Findings

A finding is a statement that some states of a variable are impossible. A finding be represented as a table of zeros and ones with a zero at the places for imposs states.

A finding on a variable A is entered into a clique V containing A by multiply Vs table by the table for the finding.

Absorption in junction trees

Definition. Let V and W be neighbours in a junction tree, let S be their separa and let $\mathbf{t}_V$, $\mathbf{t}_W$ and $\mathbf{t}_S$ be their tables. The operation *absorption* is the result of following procedure:

- calculate $\mathbf{t}_S^* = \sum_{V \setminus S} \mathbf{t}_V$;
- give S the table $\mathbf{t}_S^*$;
- give W the table $\mathbf{t}_W^* = \mathbf{t}_W \frac{\mathbf{t}_S^*}{\mathbf{t}_S}$.

We then say that W has *absorbed* from V. (See Fig. 4.25.)

HUGIN propagation

An arbitrary clique Rt in the junction tree is chosen as a root. The operati *CollectEvidence* is called in Rt followed by a call of *DistributeEvidence* in Rt.

CollectEvidence(Rt) asks all neighbours to *CollectEvidence* and they proc down the tree recursively. When all the called neighbours have finished, Rt sorbs from them.

DistributeEvidence(Rt) makes all its neighbours absorb from Rt, and afterw recursively *DistributeEvidence* to its neighbours (except Rt). See Figure 4.26.

Correctness of HUGIN propagation

Theorem 4.8 Let BN be a Bayesian network representing $P(U)$, and let T junction tree corresponding to BN. Let $e = \{f_1, \ldots, f_m\}$ be findings on the vari $\{A_1, \ldots, A_m\}$. For each i find a node containing A_i and multiply its table wi

Then, after a full round of message passing we have for each node V and sep S that

$$\mathbf{t}_V = P(V, e) \qquad \mathbf{t}_S = P(S, e) \qquad P(e) = \sum_V \mathbf{t}_V.$$

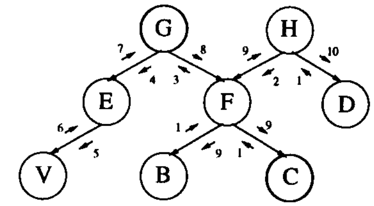

Figure 4.26 Updating through *CollectEvidence*(V) followed by *DistributeEvidence*(V).

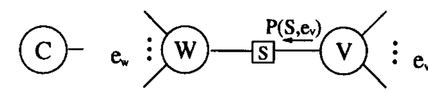

Figure 4.27 Evidence e_V has been entered at the righthand side of S. e_W has been entered at the lefthand side of S. C is used as a root for the propagation.

Side effect of Hugin Propagation

Let Rt be the root for HUGIN propagation, and let W and V be neighbours with separator S. Assume that W is closer to Rt than V. Then S divides the entered evidence in e_V and e_W (see Fig. 4.27).

A call of *CollectEvidence*(Rt) results in the table $P(S, e_V)$ being communicated from V to S. By marginalization you can calculate $P(e_V)$.

4.8 Bibliographical notes

A version of probability updating in singly connected DAGs through message passing was presented by Kim & Pearl (1983). HUGIN propagation was proposed by Jensen et al. (1990). It is a modification of an algorithm proposed by Lauritzen & Spiegelhalter (1988). Similar methods were used for pedigree analysis by Cannings et al. (1978). Shafer & Shenoy (1990) propose a different message-passing method for junction trees. Other propagation methods for multiply connected DAGs exist, e.g. arch reversal proposed by Shachter (1986) or conditioning proposed by Pearl (1986a).

The concepts of triangulated graphs and junction trees have been discovered and rediscovered with various names. In Bertele & Brioschi (1972) they are used for dynamic programming, and Beeri et al. (1983) use them for data base management. A good reference on triangulated graphs is Golumbic (1980). Tarjan & Yannakakis (1984) gives various triangulation methods and very efficient methods for testing whether a graph is triangulated. Jensen & Jensen (1994) contains a proof of Theorem 4.10 together with a method for constructing optimal junction trees from triangulated graphs.

Forward sampling was proposed by Henrion (1988). Gibbs sampling was originally introduced for image restoration by Geman & Geman (1984). Further readings on sampling methods could be Geyer (1992), Fung & Favero (1994), and Jensen et al. (1995). Gilks et al. (1994) have developed a system, BUGS, for Gibbs sampling in Bayesian networks.

Exercises

Exercise 4.1 For Table 4.6, calculate $\mathbf{t}_V \mathbf{t}_W$ and $\frac{\mathbf{t}_W}{\mathbf{t}_V}$.

Table 4.6 Table for Exercise 4.1.

	a_1	a_2	a_3
b_1	1	2	3
b_2	3	2	1

$\mathbf{t}_V$

	c_1	c_2	c_3
b_1	6	12	24
b_2	18	6	12

$\mathbf{t}_W$.

Exercise 4.2 For the universe U over the ternary variables (A, B, C) with the joint probability Table 4.7 we get the findings f_1: "A is in state a_1", and f_2: "C is in state c_1 or c_3".

Table 4.7 Table for Exercise 4.2.

	a_1	a_2	a_3
b_1	(2,4,3)	(1,4,8)	(5,0,7)
b_2	(5,10,4)	(2,3,3)	(1,5,4)
b_3	(1,5,6)	(3,3,3)	(0,6,2)

$P(A, B, C)$ multiplied by ten.

Calculate $P(B \mid f_1, f_2)$, $P(C \mid f_1, f_2)$, $P(f_1)$, $P(f_2)$ and $P(f_1, f_2)$.

Exercise 4.3 Prove that the anarchistic message passing algorithm formulated in Section 4.3.2 never runs into a deadlock: as long as there are unused message channels at least one variable can send a message. (Hint. Induction on the number of nodes and the fact that any sending sequence must start with a leaf sending.)

Exercise 4.4 Let B be independent of C given A, and let $P(A, B)$ and $P(A, C)$ be consistent. What is $P(A, B, C)$?

Exercise 4.5 Prove that a call of *CollectEvidence* in any node followed by a call of *DistributeEvidence* in the same node will result in a full propagation (all messages passed and passed when permitted).

Exercise 4.6 Construct the moral graph and a junction tree for the singly connected DAG below.

Exercise 4.7 Show that a consistent junction tree is globally consistent.

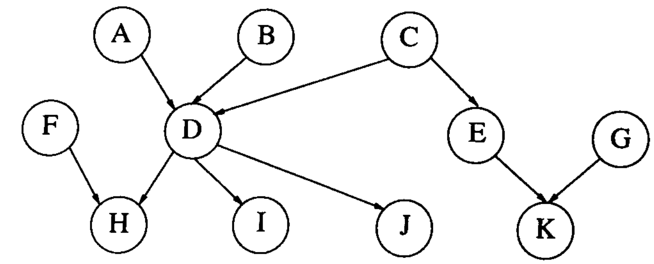

Figure for Exercise 4.6.

Exercise 4.8 (Construction of a junction tree from an elimination sequence.)
G is a triangulated graph over U, and $A_1, \ldots, A_n$ is an elimination sequence of U. C_i is the set of variables containing A_i and all its neighbours at the time of elimination.

(i) Show that each clique of G is a C_i for some i.

(ii) Show that for all $i < n$ there is a $j > i$ such that $C_i \setminus \{A_i\} \subseteq C_j$.

(iii) Assume that C_i and C_j are cliques $(i < j)$ such that $C_i \setminus \{A_i\} \subseteq C_j$. Show that there exists a junction tree for G with the link (C_i, C_j).

(iv) Use (ii) and (iii) to construct a junction tree for the graph in Figure 4.20(a).

Exercise 4.9 (i) Construct a junction tree for the DAG given below, by using the elimination order F, J, D, B, A, I, K, E.

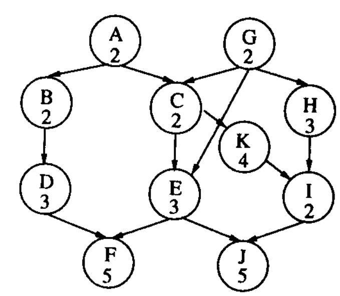

Figure for Exercise 4.9

(ii) The numbers inside the nodes indicate the number of states. Use the procedure from the end of Section 4.5 to construct a junction tree.

Exercise 4.10 (i) For the DAG given below, compute $P(A, B, C)$, when $P(A) = (0.3, 0.7)$ (see Figure and Table 4.8 for Exercise 4.10(i)).
(ii) The DAG is extended as shown in the Figure and Table 4.9 for Exercise 4.10(ii). Calculate $P(B, C, D)$.

Table 4.8 Table for Exercise 4.10(i).

	$A = y$	$A = n$
$B = y$	0.2	0.5
$B = n$	0.8	0.5
	$P(B \mid A)$	

	$A = y$	$A = n$
$C = y$	0.9	0.4
$C = n$	0.1	0.6
	$P(C \mid A)$	

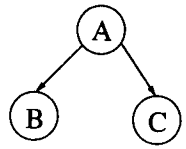

Figure for Exercise 4.10(i).

(iv) We are told that $A = y$ and $D = n$. What is $P(B)$?
(v) Initially, what was $P(A = y, D = n)$?

Exercise 4.11 (Conditioning.) Propagation methods for singly connected DAGs have existed for a long time. A propagation method for multiply connected DAGs consists of reducing a DAG to a set of singly connected DAGs.

(i) Consider the DAG (a) below with $P(A)$, $P(B \mid A)$, $P(C \mid A)$ and $P(D \mid B, C)$ given. Assume that $A = a$. Show that the DAG is reduced to the DAG (b) with $P(B \mid a)$, $P(C \mid a)$, and $P(D \mid B, C)$ given.

(ii) Show that $P(D, a) = P(D \mid b, c)P(B \mid a)P(C \mid a)$.

(iii) Assume that for all states a of A we have a reduced DAG as in (i) . Let evidence e be entered and propagated in all the reduced DAGs, yielding $P(B, e \mid a)$, $P(C, e \mid a)$, $P(D, e \mid a)$ for all a. Calculate $P(B, e)$ and $P(A, e)$.

The procedure above is called *conditioning on A*.

(iv) Reduce the DAG by conditioning on B. Show that the tables are $P(A \mid b)$, $P(C \mid A)$ and $P(D \mid C, b)$.

(v) Show that conditioning on D does not result in a singly connected DAG.

Conditioning over several variables can be performed stepwise.

(vi) Determine a minimal set of conditioning variables for the DAG given below to reduce it to singly connected DAGs.

(vii) The numbers attached to the variables indicate the number of states. Determine a conditioning resulting in a minimal number of singly connected DAGs.

Table 4.9 Table for Exercise 4.10(ii).

	$B = y$	$B = n$
$C = y$	(0, 1)	(0.7, 0.3)
$C = n$	(0.4, 0.6)	(0.5, 0.5)
	$P(D \mid B, C)$	

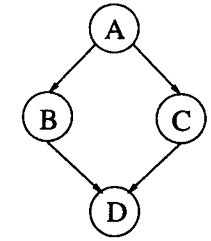

Figure for Exercise 4.10(ii).

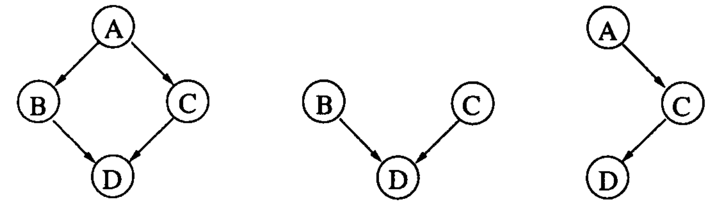

Figure for Exercise 4.11(i)–(v).

Exercise 4.12 Calculate the marginals from the sample in Table 4.5, and compare the result with the exact marginals.

Exercise 4.13 The binary variables A and B are parents of the binary variable C. $P(A) = P(B) = (0.5, 0.5)$, and the conditional probability table is an *exclusive or table*: $C = y$ if and only if exactly one of A and B is in the state y.

Show that Gibbs sampling on this structure will give either $P(C = y) = 1$ or $P(C = n) = 1$.

Exercise 4.14 Given a Bayesian network over U with evidence e entered, show that it is NP-hard to find a configuration U^* such that $P(U^*, e) > 0$.
(Hint. Look at Exercise 3.16.)

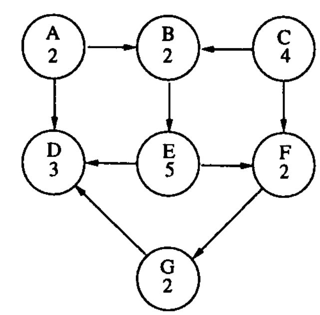

Figure for Exercise 4.11(vi)–(vii).