1228 Probabilistic Graphical Models
H. Zhu and J. Boedecker
May 1, 2026

Bayesian Networks

Contents

1 Representation

A Bayesian network represents the joint distribution of a set of discrete random variables, $X_1, \ldots, X_n$, as a *directed acyclic graph* (DAG) and some sets of conditional probabilities. Each node, that corresponds to a variable, has an associated set of conditional probabilities that contains the probability of each instance of the variable given its parents in the graph, which is known as the *conditional probability table*. The structure of the network implies a set of conditional independence assertions, which give power to this representation. Figure 1 depicts an example of a simple Bayesian network. In this example, X_4 is conditionally independent of X_1, X_3, X_5, X_6 given X_2, that is:

$$\mathbf{P}(X_4 \mid X_1, X_2, X_3, X_5, X_6) = \mathbf{P}(X_4 \mid X_2).$$

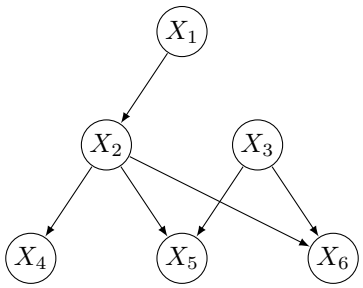


Figure 1 An example Bayesian network.

1.1 Structure

1.1.1 Mappings

Given a probability distribution $\mathbf{P}$ of $X = (X_1, \ldots, X_n)$, and its graphical representation G, there must be a correspondence between the conditional independence in $\mathbf{P}$ and in G; this is called a *mapping*. There are three basic types of mappings:

- *D-map*: all the conditional independence relations in $\mathbf{P}$ are satisfied in G.
- *I-map*: all the conditional independence relations in G are true in $\mathbf{P}$.
- *P-map*: or perfect map, it is a D-map and an I-map.

Particularly, we say the graph G and probability distribution $\mathbf{P}$ are *compatible* if G is an I-map of $\mathbf{P}$. In general, it is not always possible to have a perfect mapping of the independence relations between the graph G and the distribution $\mathbf{P}$, so we settle for what is called a *minimal I-map*: all the conditional independence relations implied by G are true in P, and if any arc is deleted in G this condition is lost.

1.1.2 d-separation

Consider three disjoint sets of variables, X, Y, and Z, which are represented as nodes in a directed acyclic graph G. To test whether X is independent of Y given Z in any distribution compatible with G, we need to test whether the nodes corresponding to variables Z 'block' all paths from nodes in X to nodes in Y. By path we mean a sequence of consecutive edges (of any directionality) in the graph, and blocking is to be interpreted as stopping the flow of information (or of dependency) between the variables that are connected by such paths. A path p is said to be *d-separated* (or blocked) by a set of nodes Z if and only if

1. the path p contains a chain $i \to m \to j$ or a fork $i \leftarrow m \to j$ such that the middle node m is in Z, or

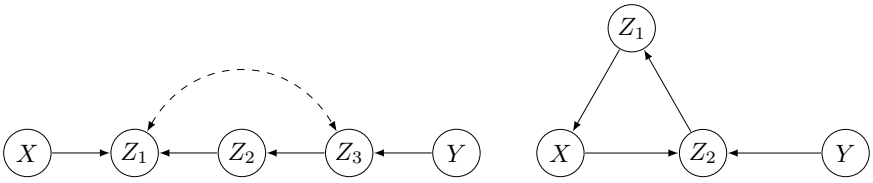


Figure 2 Graphs illustrating d-separation. In the left figure, X and Y are d-separated given Z_2 and d-connected given Z_1. In the right figure, X and Y cannot be d-separated by any set of nodes.

2. the path p contains an inverted fork (or collider) $i \to m \leftarrow j$ such that the middle node m is not in Z and such that no descendant of m is in Z.

A set Z is said to d-separate X from Y if and only if Z blocks every path from a node in X to a node in Y. In this case, random variable X is independent of Y conditional on Z in every distribution compatible with G. Conversely, if X and Y are not d-separated by Z in a directed acyclic graph G, then X and Y are dependent conditional on Z in at least one distribution compatible with G.

The intuition behind d-separation is simple and can best be recognized if we attribute causal meaning to the arrows in the graph. In causal chains $i \to m \to j$ and causal forks $i \leftarrow m \to j$, the two extreme variables are marginally dependent but become independent of each other (*i.e.*, blocked) once we condition on (*i.e.*, know the value of) the middle variable. Figuratively, conditioning on m appears to 'block' the flow of information along the path, since learning about i has no effect on the probability of j, given m. Inverted forks $i \to m \leftarrow j$, representing two causes having a common effect, act the opposite way; if the two extreme variables are (marginally) independent, they will become dependent (*i.e.*, connected through unblocked path) once we condition on the middle variable (*i.e.*, the common effect) or any of its descendants.

Example. *d-separation.* The left graph in figure 2 contains a bidirected arc $Z_1 \longleftrightarrow Z_3$, and the right graph involves a directed cycle $X \to Z_2 \to Z_1 \to X$. In the left graph of figure 2, the two paths between X and Y are blocked when none of $\{Z_1, Z_2, Z_3\}$ is measured. However, the path $X \to Z_1 \longleftrightarrow Z_3 \leftarrow Y$ becomes unblocked when Z_1 is measured. This is so because Z_1 unblocks the 'colliders' at both Z_1 and Z_3; the first because Z_1 is the collision node of the collider, the second because Z_1 is a descendant of the collision node Z_3 through the path $Z_1 \leftarrow Z_2 \leftarrow Z_3$. In the right graph of figure 2, X and Y cannot be d-separated by any set of nodes, including the empty set. If we condition on Z_2, we block the path $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ yet unblock the path $X \to Z_2 \leftarrow Y$. If we condition on Z_1, we again block the path $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ and unblock the path $X \to Z_2 \leftarrow Y$ because Z_1 is a descendant of the collision node Z_2.

According to the previous definition of d-separation, any node X is conditionally independent of all nodes in G that are not descendants of X given its parents in

the graph, $\mathbf{pa}(X)$, which is known as the *Markov assumption*. The set of parents of a variable X is called the *contour* of X. Besides, we call the *Markov blanket* of a node X, denoted as $\mathbf{mb}(X)$, is a set of nodes that make X independent of all the other nodes in G:

$$\mathbf{P}(X \mid G_{-X}) = \mathbf{P}(X \mid \mathbf{mb}(X)).$$

For a Bayesian network, the Markov blanket of X consists of:

- the parents of X and,
- the children of X and,
- the other parents of the children of X.

For instance, in the Bayesian network of figure 1, the Markov blanket of X_4 is X_2, and the Markov blanket of X_2 is X_1, X_3, X_4, X_5, X_6.

The structure of a Bayesian network can then be specified by the parents of each variable. For example the Bayesian network in figure 1, its structure can be specified as:

$$\left\{ \begin{array}{l} \mathbf{pa}(X_1) = \emptyset, \\ \mathbf{pa}(X_2) = \{X_1\}, \\ \mathbf{pa}(X_3) = \emptyset, \\ \mathbf{pa}(X_4) = \{X_2\}, \\ \mathbf{pa}(X_5) = \{X_2, X_3\}, \\ \mathbf{pa}(X_6) = \{X_2, X_3\} \end{array} \right\}.$$

Then using the chain rule, we can specify the joint probability distribution of the set of variables in a Bayesian network as the product of the conditional probability of each variable given its parents:

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid \mathbf{pa}(X_i)).$$

For the example in figure 1:

$$\begin{aligned} \mathbf{P}(X_1, \ldots, X_6) = \mathbf{P}(X_1)\, \mathbf{P}(X_2 \mid X_1)\, \mathbf{P}(X_3) \\ \times \mathbf{P}(X_4 \mid X_2)\, \mathbf{P}(X_5 \mid X_2, X_3)\, \mathbf{P}(X_6 \mid X_2, X_3). \end{aligned}$$

1.2 Parameters

To complete the specification of a Bayesian network, we need to define its parameters, which are the conditional probabilities of each node given its parents in the graph: $\mathbf{P}(X_i \mid \mathbf{pa}(X_i))$. In the case of continuous variables, wee need to specify a function that relates the density function of each variable to the density of its parents; in the case of discrete variables, the number of parameters to specify $\mathbf{P}(X_i \mid \mathbf{pa}(X_i))$ can be combinatorially large as the number of parents of X_i increases. Two main alternatives have been proposed to overcome this issue, one is based on *canonical models* and the other on graphical representations of conditional probability tables.

1.2.1 Canonical models

Canonical models represent the relations between a set of random variables for particular interactions using few parameters. It can be applied when the probabilities of a random variable in a Bayesian network conform to certain canonical relations with respect to the configurations of its parents. There are several classes of canonical models, the most common are the *noisy-OR* and *noisy-AND* for binary variables, and their extensions for multivalued variables, *noisy-max* and *noisy-min*, respectively. Canonical models can provide a considerable reduction in the number of parameters when a variable has many parents; and also some inference techniques take advantage of this compact representation.

Example. *Noisy-OR.* Consider an OR logic gate, in which the output is true if any of its inputs are true. The noisy-OR model is based on the concept of the logic OR; the difference is that there is a certain (small) probability that the variable is not true even if one or more of its parents are true. The noisy-OR model is applied when several variables or causes can produce an effect if any one of them is true, and as more of the causes are true, the probability of the effect increases. For instance, the effect could be a certain symptom or effect, E, and the causes are a number of possible diseases, $C_1, \ldots, C_n$, that can produce the symptom, such that if none of the diseases is present (all false), the symptom does not appear; and when any disease is present (true) the symptom is present with high probability and it increases as the number of C_i = true increases. A graphical representation of a noisy-OR relation in a Bayesian network is depicted in figure 3.

Formally, the following two conditions must be satisfied for a noisy-OR canonical model to be applicable:

- *Independence of exceptions*: if an effect is the manifestation of several causes, the mechanisms that inhibit the occurrence of the effect, E = false, under one cause are independent of the mechanisms that inhibit it under the other causes[1], *i.e.*,

$$\mathbf{P}(E = \text{false} \mid C_1, \ldots, C_n) = \prod_{i=1}^{n} \mathbf{P}(E = \text{false} \mid C_i). \tag{1.1}$$

- *Responsibility*: the effect E is false if all the possible causes are false. Together with (1.1), this implies

$$\mathbf{P}(E = \text{false} \mid C_i = \text{false}) = 1, \quad i = 1, \ldots, n.$$

The probability that the effect E is inhibited (it does not occur) under cause C_i is defined as:

$$q_i = \mathbf{P}(E = \text{false} \mid C_i = \text{true}).$$

Given this definition and the previous conditions, the parameters in the conditional probability table for a noisy-OR model can be obtained using the following expressions when all the n causes are true:

$$\mathbf{P}(E = \text{false} \mid C_1 = \text{true}, \ldots, C_n = \text{true}) = \prod_{i=1}^{n} q_i,$$

[1] This independence does not necessarily hold for E = true.

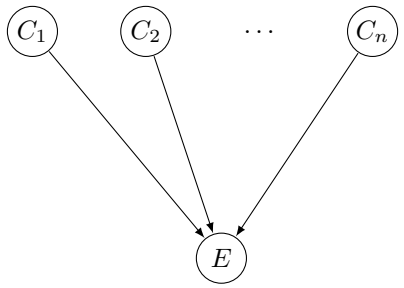


Figure 3 Graphical representation of a noisy-OR structure. The cause variables $C_1, \ldots, C_n$ are the parents of the effect variable E.

Table 1 Conditional probability table for a noisy-OR variable with three parents and parameters $q_1 = q_2 = q_3 = 0.1$.

C_1	0	0	0	0	1	1	1	1
C_2	0	0	1	1	0	0	1	1
C_3	0	1	0	1	0	1	0	1
$\mathbf{P}(E = 0)$	1	0.1	0.1	0.01	0.1	0.01	0.01	0.001
$\mathbf{P}(E = 1)$	0	0.9	0.9	0.99	0.9	0.99	0.99	0.999

and

$$\mathbf{P}(E = \text{true} \mid C_1 = \text{true}, \ldots, C_n = \text{true}) = 1 - \prod_{i=1}^{n} q_i.$$

In general, if k out of n causes are true, then (informally):

$$\mathbf{P}(E = \text{false} \mid C_1, \ldots, C_n) = \prod_{i=1}^{k} q_i,$$

so that if all the causes are false then the effect is false with probability one. Thus, only one parameter is required per parent variable to construct the conditional probability table, which is the inhibition probability q_i. In this case the number of independent parameters $(q_1, q_2, \ldots, q_n)$ increases linearly with the number of parents, instead of exponentially. As an example, consider a noisy-OR model with 3 causes, C_1, C_2, and C_3, where the inhibition probabilities are the same for the three, $q_1 = q_2 = q_3 = 0.1$. Given these parameters we can obtain the conditional probability table for the effect variable E, as shown in table 1.

1.2.2 Graphical representations

Canonical models apply in certain situations but do not provide a general solution for compact representations of conditional probability tables. An alternative

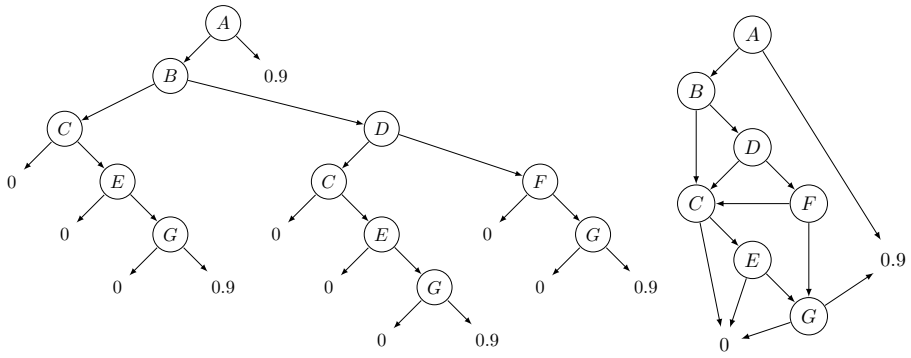


Figure 4 Examples about decision tree (*left*) and decision diagram (*right*) representation of a conditional probability table.

representation is based on the observation that within each conditional probability table, the same probability values tend to be repeated several times. Thus, it is not necessary to represent these repeated values many times. A representation that takes advantage of this condition is *decision tree*, which could be used for representing a conditional probability table in a compact way. In a decision tree, each internal node corresponds to a variable in the conditional probability table, and the branches from a node correspond to the different values that a variable can take. The leaf nodes in the tree represent the different probability values. A trajectory from the root to a leaf, specifies a probability value for the corresponding variables. If a variable is omitted in a trajectory, it means that the conditional probability table has the same probability for all values of this variable. Another graphical representation of conditional probability tables is *decision diagram*, which extends decision tree by considering a directed acyclic graph structure, such that it is not restricted to a tree. This avoids the need to duplicate repeated probability values in the leaf nodes, and in some cases provides an even more compact representation. Examples about these two graphical representations of conditional probability tables, $\mathbf{P}(X \mid A, B, C, D, E, F, G)$, is shown in figure 4, assuming variables A, B, C, D, E, F, G are all binary.

2 Inference

2.1 Belief propagation

The *belief propagation* algorithm only applies to *singly connected graphs* (trees and polytrees). Although there is an extension of belief propagation for general Bayesian networks, but the convergence is not guaranteed.

Given certain evidence E, the posterior probability for any variable $X = x_i$ can

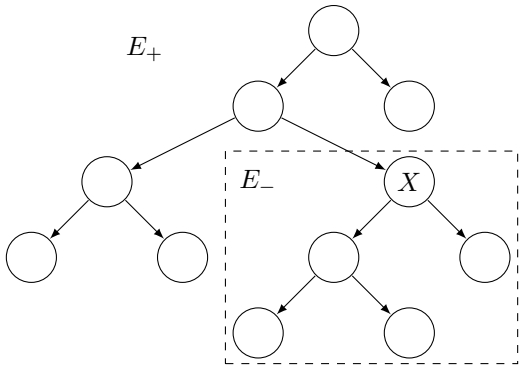


Figure 5 Divide the evidence E into two independent components E_+ and E_- with node X.

be obtained by applying the Bayes rule:

$$\mathbf{P}(x_i \mid E) = \frac{\mathbf{P}(E \mid x_i)\,\mathbf{P}(x_i)}{\mathbf{P}(E)}. \tag{2.1}$$

Given that the Bayesian network has a tree structure, the network can then be divided by any node into two independent subtrees. Thus, we can separate the evidence into:

- E_-: evidence of the rooted tree in X, and
- E_+: all other evidence,

as shown in figure 5. Since E_+ and E_- are conditionally independent given X, thus

$$\begin{aligned}
\mathbf{P}(x_i \mid E) &= \frac{\mathbf{P}(E \mid x_i)\,\mathbf{P}(x_i)}{\mathbf{P}(E)} && \text{(from (2.1))} \\
&= \frac{\mathbf{P}(E_-, E_+ \mid x_i)\,\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_- \mid x_i)\,\mathbf{P}(E_+ \mid x_i)\,\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_- \mid x_i)\,\mathbf{P}(x_i \mid E_+)\,\mathbf{P}(E_+)\cancel{\mathbf{P}(x_i)}}{\mathbf{P}(E)\cancel{\mathbf{P}(x_i)}} \\
&= \frac{1}{Z}\,\mathbf{P}(x_i \mid E_+)\,\mathbf{P}(E_- \mid x_i), && (2.2)
\end{aligned}$$

where $\frac{1}{Z} = \frac{\mathbf{P}(E_+)}{\mathbf{P}(E)}$ is the normalization constant. Let us introduce two auxiliary variables:

$$\mu(x_i) = \mathbf{P}(x_i \mid E_+),$$

and

$$\lambda(x_i) = \mathbf{P}(E_- \mid x_i),$$

then (2.2) can be written as

$$\mathbf{P}(x_i \mid E) = \frac{1}{Z}\mu(x_i)\lambda(x_i). \tag{2.3}$$

The expression (2.3) is the basis of the belief propagation algorithm to obtain the posterior probability of all non-instantiated nodes. The computation of the posterior probability of any node X is decomposed into two parts: (1) the evidence λ coming from the children of X in the tree, and the evidence μ coming from the parent of X. We can think of each node X in the tree as a simple processor that stores its vectors

$$\mu(X) = (\ldots, \mathbf{P}(x_i \mid E_+), \ldots), \quad i = 1, 2, \ldots$$

and

$$\lambda(X) = (\ldots, \mathbf{P}(E_- \mid x_i), \ldots), \quad i = 1, 2, \ldots,$$

and its conditional probability table $\mathbf{P}(X \mid \mathbf{pa}(X))$. The evidence is propagated via a message passing mechanism, in which each node sends the corresponding messages to its parent and children in the tree.

Next we derive the equations for the messages. For the λ messages, given that the children of X are conditionally independent given x_i, we have

$$\lambda(x_i) = \mathbf{P}(E_- \mid x_i) = \prod_k \mathbf{P}(E_-^{(k)} \mid x_i),$$

where $E_-^{(k)}$ is the evidence coming from the tree rooted in the kth child $Y^{(k)}$ of X. Applying the rule of total probability conditioning on $Y^{(k)}$, we obtain

$$\begin{aligned}
\mathbf{P}(E_-^{(k)} \mid x_i) &= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid x_i, y^{(k)})\, \mathbf{P}(y^{(k)} \mid x_i) \\
&= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid y^{(k)})\, \mathbf{P}(y^{(k)} \mid x_i) \\
&= \sum_{y^{(k)} \in Y^{(k)}} \lambda(y^{(k)})\, \mathbf{P}(y^{(k)} \mid x_i),
\end{aligned} \tag{2.4}$$

where the second equivalence comes from the fact that the evidence coming from the tree rooted in node $Y^{(k)}$ is conditionally independent of X given $Y^{(k)}$. Hence the λ evidence for each note can be calculated recursively according to the λ evidence of its children by performing a bottom-up propagation, which is shown in the left of figure 6.

Similarly, for the μ messages, we apply the rule of total probability conditioning on the parent node $W = \mathbf{pa}(X)$ of X:

$$\begin{aligned}
\mu(x_i) &= \mathbf{P}(x_i \mid E_+) \\
&= \sum_{w \in W} \mathbf{P}(x_i \mid E_+, w)\, \mathbf{P}(w \mid E_+)
\end{aligned}$$

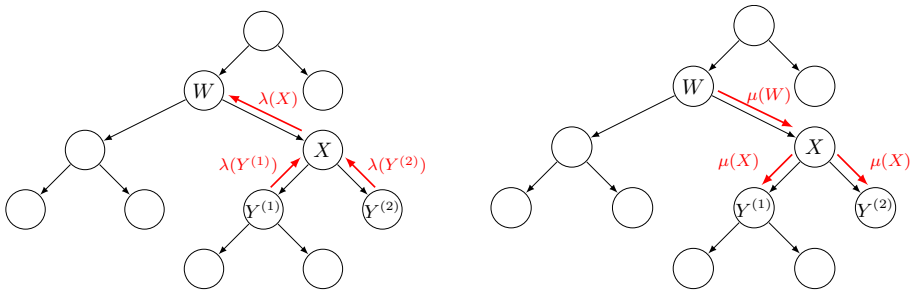


Figure 6 *Left.* Bottom-up propagation of the λ evidence. *Right.* Top-down propagation of the μ evidence.

$$= \sum_{w \in W} \mathbf{P}(x_i \mid w)\, \mathbf{P}(w \mid E_+), \tag{2.5}$$

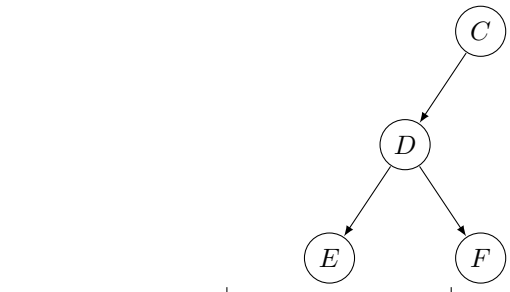
where the third equivalence is, again, because of X is conditionally independent of the evidence E_+ given W. $\mathbf{P}(w \mid E_+)$ corresponding to the probability of w given the evidence coming from all the tree except the subtree rooted on X. According to (2.3) to (2.4), it can be written as

$$\begin{aligned} \mathbf{P}(w \mid E_+) &= \frac{1}{Z}\mu(w) \prod_{E^{(k)}_{W-} \neq E_-} \mathbf{P}(E^{(k)}_{W-} \mid w) \\ &= \frac{1}{Z}\mu(w) \prod_{X^{(k)} \neq X} \sum_{x^{(k)} \in X^{(k)}} \lambda(x^{(k)})\, \mathbf{P}(x^{(k)} \mid w), \end{aligned} \tag{2.6}$$

where $X^{(k)}$ is the kth child of W, and $E^{(k)}_{W-}$ is the evidence coming from the tree rooted in $X^{(k)}$. Hence, the μ evidence of each node can be recursively expressed as a function of the μ evidence of its parent W, and the λ evidence of all other children of W. To calculate the μ evidence for all nodes, we can first perform a bottom-up propagation from all leaves of the tree (figure 6, left) to obtain the λ evidence of all nodes, then perform a top-down propagation from the root of the tree (figure 6, right).

Before the propagation starts, we should assign the evidence to those instantiated variables, and define the prior of the root and leaf nodes of the tree. For example, we can consider a uniform distribution $\lambda = \mathbf{1}$ if a leaf node is unknown, and a one-hot coding for a known leaf node (one for the assigned value and zero for all other values). After obtaining the λ and μ evidence vector of all nodes in the tree from the propagation, the posterior probability of any variable X is obtained by combining these vectors using (2.3) and normalizing.

Example. *Belief propagation.* We now illustrate the belief propagation algorithm with a simple example. Consider the Bayesian network in figure 7 with 4 binary variables C, D, E, F, and the only evidence is $E = e_1$. Then the initial conditions for the



c_1	c_2
0.8	0.2

	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

Figure 7 A simple Bayesian network with corresponding conditional probability table.

leaf nodes are: $\lambda(E) = (1, 0)$ and $\lambda(F) = (1, 1)$. Propagating to the parent node D is basically multiplying the λ vectors by the corresponding conditional probability tables:

$$\begin{aligned}\lambda(D) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix} \\ &= \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}.\end{aligned}$$

Now propagating it to its parent C:

$$\lambda(C) = \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.7 \\ 0.1 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix}.$$

In this way, we complete the bottom-up propagation.

We now perform the top-down propagation. Given that C is not instantiated with prior $(0.8, 0.2)$, we define $\mu(C) = (0.8, 0.2)$ and propagate to its child D:

$$\mu(D) = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}.$$

We now propagate to its child F; however, given that D has another child, E,we also need to consider the λ message from this other child, thus:

$$\mu(F) = \left(\begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \right)^T \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} = \begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix}.$$

This completes the top-down propagation. Given the λ and μ vectors for each unknown variable, we just multiply them term by term and then normalize to obtain

the posterior probabilities:

$$\mathbf{P}(C) = \frac{1}{Z}\begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \odot \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.69 \\ 0.16 \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.19 \end{bmatrix},$$

$$\mathbf{P}(D) = \frac{1}{Z}\begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.77 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 0.92 \\ 0.08 \end{bmatrix},$$

$$\mathbf{P}(F) = \frac{1}{Z}\begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 0.32 \end{bmatrix}.$$

2.2 Variable elimination

Assume a Bayesian network representing the joint probability distribution of $X = \{X_1, \ldots, X_n\}$. We want to calculate the posterior probability of a certain variable of subset of variables X_H, given a subset of evidence variables X_E; the remaining variables are X_R, such that $X = \{X_H \cup X_E \cup X_R\}$. The posterior probability of X_H given the evidence is:

$$\mathbf{P}(X_H \mid X_E) = \frac{\mathbf{P}(X_H, X_E)}{\mathbf{P}(X_E)}.$$

We can obtain both terms via marginalization of the joint distribution:

$$\mathbf{P}(X_H, X_E) = \sum_{X_R} \mathbf{P}(X),$$

and

$$\mathbf{P}(X_E) = \sum_{X_H} \mathbf{P}(X_H, X_E).$$

The objective of the variable elimination method is to perform these calculations efficiently. To achieve this, we can first represent the joint distribution as a product of local probabilities according to the network structure. Then, summations can be carried out only on the subset of terms which are a function of the variables being normalized. This approach takes advantage of the properties of summation and multiplication, resulting in the number of necessary operations being reduced.

Example. *Variable elimination.* Consider the Bayesian network in figure 8, where we want to obtain $\mathbf{P}(A \mid D)$. In order to achieve this we need to obtain $\mathbf{P}(A, D)$ and $\mathbf{P}(D)$. To calculate the first term we should eliminate B, C, and E from the joint distribution, that is:

$$\begin{aligned}\mathbf{P}(A, D) &= \sum_B \sum_C \sum_E \mathbf{P}(A, B, C, D, E) \\ &= \sum_B \sum_C \sum_E \mathbf{P}(A)\,\mathbf{P}(B \mid A)\,\mathbf{P}(C \mid A)\,\mathbf{P}(D \mid B, C)\,\mathbf{P}(E \mid C)\end{aligned}$$

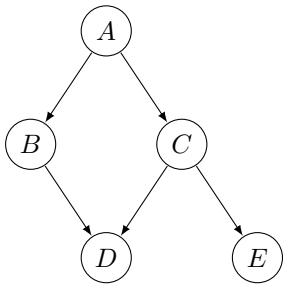


Figure 8 A simple Bayesian network.

$$= \mathbf{P}(A) \sum_{B} \left(\mathbf{P}(B \mid A) \sum_{C} \left(\mathbf{P}(C \mid A)\, \mathbf{P}(D \mid B, C) \sum_{E} \mathbf{P}(E \mid C) \right) \right).$$

If we consider that all variables are binary, this implies a reduction from 32 operations to 15 operations.

The critical aspect of the variable elimination algorithm is to select the appropriate order for eliminating each variable, as this has an important effect on the number of required operations. The different terms that are generated during the calculations are known as *factors* which are functions over a subset of variables, that map each instantiation of these variables to a non-negative number (not necessarily probabilities). In general a factor can be represented as $f(X_1, \ldots, X_m)$. For instance, in the previous example, one of the factors is $f(C, E) = \mathbf{P}(E \mid C)$, which is a function of two variables. The computational complexity in terms of space and time of the variable elimination algorithm is determined by the size of the factors, *i.e.*, the number of variables on which the factor is defined. Basically, the complexity for eliminating any number of variables is exponential on the number of variables in the factor. Thus, the order in which the variables are eliminated should be selected so that the largest factor is kept to a minimum.

In general, finding the best order of variable elimination is NP-hard, but there are several heuristics that help to determine a good ordering for variable elimination. These heuristics can be explained based on the *interaction graph*, which is an undirected graph that is built during the process of variable elimination. The variables of each factor form a clique in the interaction graph. The initial interaction graph is obtained from the original Bayesian network structure by eliminating the direction of the arcs, and adding additional arcs between each pair of non-connected variables that have a common child. Then, each time a variable X_j is eliminated, the interaction graph is modified by adding an arc between each pair of neighbors of X_j that are not connected, and deleting variable X_j from the graph. For example, the interaction graphs that result from the Bayesian network in figure 8 by the following elimination ordering: E, D, C, B, is depicted in figure 9. Two popular heuristics for determining the elimination ordering, which can be obtained from the interaction graph, are the following:

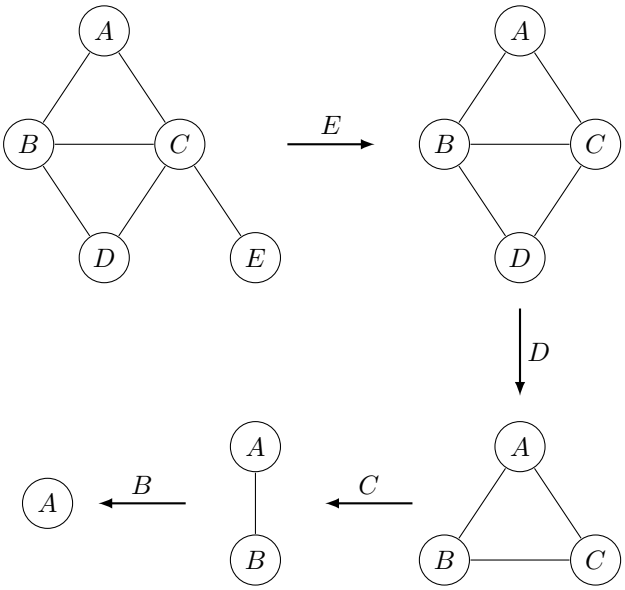


Figure 9 Interaction graphs resulting from the elimination of variables with ordering E, D, C, B from the Bayesian network in figure 8.

- *Min-degree*: eliminate the variable that leads to the smallest possible factor, which is equivalent to eliminating the variable with the smallest number of neighbors in the current interaction graph.
- *Min-fill*: eliminate the variable that leads to adding the minimum number of edges to the interaction graph.

A disadvantage of variable elimination is that it only obtains the posterior probability of one variable (or subset of variables). To obtain the posterior probability of each non-instantiated variable in a Bayesian network, the calculations have to be repeated for each variable.

2.3 Conditioning

The conditioning method is based on the fact that an instantiated variable blocks the propagation of the evidence in a Bayesian network. Thus, we can cut the graph at an instantiated variable, and this can transform a multi-connected graph into a polytree, for which we can apply the belief propagation algorithm introduced in §2.1. In general, a subset of variables can be instantiated to transform a multi-connected network into a singly connected graph. If these variables are not actually known, we can set them to each of their possible values, and then do probability propagation for each value. With each propagation we obtain the posterior probabilities for all unknown variables. Then, the final probability values are obtained as a weighted combination of these posterior probabilities.

First we will introduce the conditioning algorithm assuming we only need to partition a single variable and then we will extend it for multiple variables. Formally, we want to obtain the probability of any variable X, given the evidence E, conditioning on variable A. The variable A is selected such that by instantiating A, the multi-connected graph can be transformed to a singly connected graph. By the rule of total probability, we have

$$\mathbf{P}(X \mid E) = \sum_{a \in A} \mathbf{P}(X \mid E, a)\, \mathbf{P}(a \mid E), \tag{2.7}$$

where $\mathbf{P}(X \mid E, a)$ is the posterior probability of X which is obtained by probability propagation for each possible value of A, and the probability $\mathbf{P}(a \mid E)$ is the weight. By applying the Bayes rule, we obtain the following equation to estimate the weight:

$$\mathbf{P}(a \mid E) = \frac{1}{Z}\, \mathbf{P}(a)\, \mathbf{P}(E \mid a), \tag{2.8}$$

for all $a \in A$, where $\frac{1}{Z}$ is the normalization constant. The first term $\mathbf{P}(a)$ can be obtained by propagating without evidence. The second term $\mathbf{P}(E \mid a)$ can be calculated by propagation with $A = a$ to obtain the probability of the evidence variables.

Example. *Conditioning.* Consider the Bayesian network in figure 8, this multi-connected network can be transformed into a polytree by assuming A is instantiated, as shown in figure 10. If the evidence is D and E, then probabilities of the other variables A, B, and C can be obtained via conditioning following these steps:

1. Obtain the prior probability of A.
2. Obtain the probability of the evidence nodes D and E for each value of A by probability propagation in the polytree.
3. Calculate the weights $P(a \mid D, E)$ according to (2.8), with the probabilities obtained from the last two steps.
4. Estimate the probability of B and C for each value of A given the evidence by probability propagation in the polytree.
5. Obtain the posterior probabilities for B and C from the last two steps by applying (2.7).

In general, if we need to instantiate n variables to transform a multi-connected Bayesian network to a polytree, propagation must be performed for all the combinations of values of the instantiated variables. If each variable has k values, the number of propagations is k^n. The procedure is basically the same as described above for one variable with increased complexity.

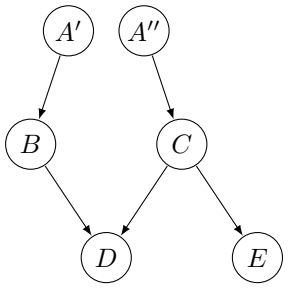


Figure 10 Transform the Bayesian network in figure 8 into a singly connected network by instantiating A.

2.4 Junction tree algorithm

The *junction tree* method is based on a transformation of the Bayesian network to a junction tree, where each node in this tree is a group or cluster of variables from the original network. Probabilistic inference is performed over this new representation. Before going into the details about the algorithm, we first introduce some background knowledge about graph theory.

2.4.1 Complete graph and cliques

A *complete graph* is a graph, in which each pair of nodes is adjacent, *i.e.*, there is an edge between each pair of nodes. A *complete set* is a subset of some graph G that induces a complete subgraph of G. It is a subset of vertices of G so that each pair of nodes in this subgraph is adjacent. A *clique* C is a subset of graph G such that it is a complete set that is maximal, *i.e.*, there is no other complete set in G that contains C.

2.4.2 Ordering and triangulation

An *ordering* of the nodes in a graph consists in assigning an integer to each node. Given a graph $G = (V, E)$, with n vertices, then $\alpha = (V_1, \ldots, V_n)$ is an ordering of the graph, where V_i is before V_j according to this ordering if $i < j$. An ordering α of a graph $G = (V, E)$ is a *perfect ordering* if all the adjacent vertices of each vertex V_i that are before V_i are completely connected according to this ordering. That is, for every node V_i, $\mathbf{adj}(V_i) \cap \{V_1, \ldots, V_{i-1}\}$ is a complete subgraph of G, where $\mathbf{adj}(V)$ denotes the adjacent nodes of V. Figure 11 depicts an example of a perfect ordering.

Consider the set of m cliques of an undirected connected graph G. In an analogous way as an ordering of the nodes, we can define an ordering of the cliques as $\beta = (C_1, \ldots, C_m)$. An ordering β of the cliques has the *running intersection property*, if all the common nodes of each clique C_i with previous cliques according to this order are contained in some clique C_j with $j < i$. That is, for every clique C_i with $i > 1$, there exists a clique C_j with $j < i$ such that $C_i \cap \{C_1, \ldots, C_{i-1}\} \subseteq C_j$.

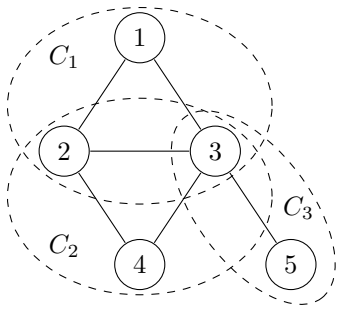


Figure 11 An example of an ordering of nodes and cliques in a graph. In this case, the nodes have a perfect ordering, and the ordering of the cliques satisfies the running intersection property.

In this case, we call the clique C_j the parent of clique C_i. It is possible that a clique has more than one parent. The cliques C_1, C_2, and C_3 in figure 11 satisfy the running intersection property. In this example, C_1 is the parent of C_2, and C_1 and C_2 are parents of C_3.

A graph G is *triangulated* if every simple circuit of length greater than three in G has a chord. A chord is an edge that connects two of the vertices in the circuit and that is not part of that circuit. For example, in the triangulated graph shown in figure 11, the circuit formed by the vertices $1 \to 2 \to 4 \to 3 \to 1$ has a chord that connects nodes 2 and 3. Given a graph G, satisfying the triangulation property is a condition for achieving a perfect ordering of the vertices, and having an ordering of the cliques that satisfies the running intersection property.

2.4.3 Maximum cardinality search

Given that a graph is triangulated, the maximum cardinality search algorithm provides a perfect ordering of the nodes. Let $G = (V, E)$ be an undirected graph with n vertices, we first select any node from V and assign it index 1. Then from all the non-indexed vertices, select the one with higher number of adjacent indexed vertices and assign it the next number, until all nodes in G have been numbered. If there are multiple nodes with the same number of adjacent indexed vertices, we can choose from one of them randomly for the current index.

Given a perfect ordering of the vertices, it is easy to number the cliques so the order satisfies the running intersection property. For this, the cliques are numbered in inverse order. Given a set of m cliques, the clique that has the node with the highest index is assigned m, and the clique that includes the next highest indexed node is assigned $m-1$, until all cliques are numbered. This method can be illustrated with the example in figure 11. The node with the highest number is 5, so the clique that contains it is C_3. The next highest node is 4, so the clique that includes it is C_2. The remaining clique is C_1.

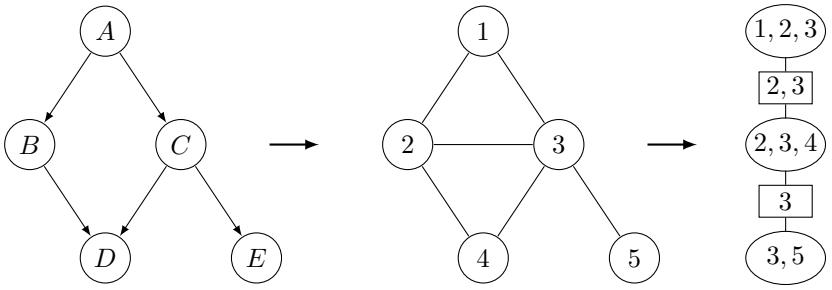


Figure 12 Transformation of the Bayesian network in figure 8 to a junction tree. The ellipse nodes represent cliques and the rectangle nodes are the separators.

2.4.4 Junction tree algorithm

The intuition behind the junction tree method is based on transforming a Bayesian network (which is a directed graph) to an undirected graph, and then clustering the variables into cliques so that the resulting graph is singly connected. Finally the belief propagation can be performed on the resulting singly connected network.

The transformation proceeds as follows:

1. Eliminate the directionality of the arcs.
2. Moralize the graph by adding an arc between pairs of nodes with common children, and add additional arcs if necessary to make the graph triangulated.
3. Order the nodes in the graph with maximum cardinality search.
4. Obtain and order the cliques of the graph such that the order satisfies the running intersection property.
5. Build a junction tree according to the clique ordering.

Figure 12 shows an example of transforming a Bayesian network to a junction tree. Those common variables of neighbor cliques in the junction tree are called separators. Given the relevance of these separators, the junction tree is usually drawn including the separator nodes, depicted as rectangles.

Once the junction tree is built, inference is based on probability propagation over the junction tree, in an analogous way as for tree-structured Bayesian networks. In practice, the belief propagation process on a junction tree is divided into two stages: preprocessing and propagation. In the preprocessing phase, the potentials ψ of each clique are obtained through the following steps:

1. Determine the set of variables for each clique C_i.
2. Determine the set of variables that are shared with the previous (parent) clique, *i.e.*, the separators S_i.

3. Determine the set of other variables R_i that are in C_i but not in S_i.
4. Calculate the potential of each clique as $\psi(C_i) = \prod_{X \in R_i} \mathbf{P}(X \mid \mathbf{pa}(X))$.

The propagation phase proceeds in a similar way to belief propagation for trees, by propagating λ messages bottom-up and μ messages top-down:

- *Bottom-up propagation.*
  1. Start from the leaf clique, calculate the λ message to send to the parent clique: $\lambda(C_i) = \sum_{R_i} \psi(C_i)$.
  2. Update the potential of each clique with the λ messages from its children: $\psi'(C_j) = \lambda(C_i)\psi(C_j)$.
  3. Repeat the previous two steps until reaching the root clique, and obtain $\mathbf{P}(C_{\text{root}}) = \psi'(C_{\text{root}})$.
- *Top-down propagation.*
  1. Start from the root clique, calculate the μ message to send to each child node C_i by its parent C_j: $\mu(C_i) = \sum_{C_j - S_i} \mathbf{P}(C_j)$.
  2. Update the potential of each clique when receiving the μ message from its parent and obtain: $\mathbf{P}(C_i) = \psi'(C_i)\frac{\mu(C_i)}{\lambda(C_i)}$.
  3. Repeat the previous two steps until reaching the leaf nodes in the junction tree.

At the end of this propagation in both directions, each clique has the joint marginal probability of the variables that conform it. Hence, the marginal posterior probabilities of each variable can be obtained from the clique via marginalization.

Example. *Junction tree algorithm.* Consider the junction tree in figure 12, the preprocessing phase is obtaining:

$C_1 = \{A, B, C\}$	$C_2 = \{B, C, D\}$	$C_3 = \{C, E\}$
$S_1 = \emptyset$	$S_2 = \{B, C\}$	$S_3 = \{C\}$
$R_1 = \{A, B, C\}$	$R_2 = \{D\}$	$R_3 = \{E\}$
$\psi(C_1) = \mathbf{P}(A)\,\mathbf{P}(B \mid A)\,\mathbf{P}(C \mid A)$	$\psi(C_2) = \mathbf{P}(D \mid B, C)$	$\psi(C_3) = \mathbf{P}(E \mid C)$.

Then in the propagation phase, first C_3 sends a λ message to C_2:

$$\lambda(C_3) = \sum_E \psi(C_3) = \sum_E \mathbf{P}(E \mid C),$$

and the potential of C_2 is updated as:

$$\psi'(C_2) = \psi(C_2)\lambda(C_3) = \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C).$$

Next, the λ message from C_2

$$\lambda(C_2) = \sum_D \psi'(C_2) = \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)$$

is sent to C_1, and the potential of C_1 is updated as:

$$\psi'(C_1) = \psi(C_1)\lambda(C_2) = \mathbf{P}(A)\,\mathbf{P}(B \mid A)\,\mathbf{P}(C \mid A) \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C).$$

Finally we assign $\mathbf{P}(C_1) = \psi'(C_1)$, which ends the bottom-up propagation. Note that the obtained probability $\mathbf{P}(C_1)$ is equivalent to the joint marginal probability of the variables in clique C_1 since

$$\begin{aligned}
\mathbf{P}(C_1) &= \psi'(C_1) \\
&= \mathbf{P}(A)\,\mathbf{P}(B \mid A)\,\mathbf{P}(C \mid A) \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C) \\
&= \sum_{D,E} \mathbf{P}(A)\,\mathbf{P}(B \mid A)\,\mathbf{P}(C \mid A)\,\mathbf{P}(D \mid B, C)\,\mathbf{P}(E \mid C) \\
&= \sum_{D,E} \mathbf{P}(A, B, C, D, E) \\
&= \mathbf{P}(A, B, C).
\end{aligned}$$

Now starts the top-down propagation. Clique C_1 sends μ message

$$\mu(C_2) = \sum_{C_1 - S_2} \mathbf{P}(C_1) = \sum_A \mathbf{P}(A, B, C)$$

to clique C_2, and we obtain

$$\begin{aligned}
\mathbf{P}(C_2) &= \psi'(C_2) \frac{\mu(C_2)}{\lambda(C_2)} \\
&= \frac{\mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C) \sum_A \mathbf{P}(A, B, C)}{\sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)} \\
&= \mathbf{P}(D \mid B, C)\,\mathbf{P}(B, C) \\
&= \mathbf{P}(B, C, D),
\end{aligned}$$

which is also equivalent to the joint marginal probability of the variables in clique C_2. Finally, clique C_2 sends μ message

$$\mu(C_3) = \sum_{C_2 - S_3} \mathbf{P}(C_2) = \sum_{B,D} \mathbf{P}(B, C, D)$$

to clique C_3, and we get

$$\begin{aligned}
\mathbf{P}(C_3) &= \psi'(C_3) \frac{\mu(C_3)}{\lambda(C_3)} \\
&= \frac{\mathbf{P}(E \mid C) \sum_{B,D} \mathbf{P}(B, C, D)}{\sum_E \mathbf{P}(E \mid C)}
\end{aligned}$$

$$= \mathbf{P}(E \mid C)\,\mathbf{P}(C)$$
$$= \mathbf{P}(C, E).$$

This ends the top-down propagation.

After obtaining the joint marginal probability of the variables for all cliques, the single variable's probabilities can be calculated via marginalization:

$$\mathbf{P}(A) = \textstyle\sum_{B,C} \mathbf{P}(C_1) \quad \mathbf{P}(B) = \sum_{A,C} \mathbf{P}(C_1) \quad \mathbf{P}(C) = \sum_{A,B} \mathbf{P}(C_1)$$
$$\mathbf{P}(D) = \textstyle\sum_{B,C} \mathbf{P}(C_2)$$
$$\mathbf{P}(E) = \textstyle\sum_{C} \mathbf{P}(C_3).$$

2.5 Sampling based methods

Stochastic simulation algorithms consist in simulating the Bayesian several times, where each simulation gives a sample value for all non-instantiated variables. Then the posterior probability of each variable is approximated in terms of the frequency of each value in the sample space. This process gives an estimate of the posterior probability which depends on the number of samples. However, the computational cost is not affected by the complexity of the network.

2.5.1 Logic sampling

Logic sampling is a basic stochastic simulation algorithm that generates samples according to the following procedure:

1. Generate sample values for all root nodes of the Bayesian network according to their prior probabilities $\mathbf{P}(X)$.
2. Generate samples for the children of the sampled nodes, according to their conditional probabilities $\mathbf{P}(X \mid \mathbf{pa}(X))$.
3. Repeat the second step until all leaf nodes are reached.

The previous procedure is repeated n times to generate n samples. Then the probability of possible values of each variable is estimated as the frequency that the value occurs in the n samples, *i.e.*,

$$\mathbf{P}(X = x_k) = \frac{1}{n} \sum_{i=1}^{n} I_{x_k}(x_i),$$

where the indicator function $I_{x_k}(x_i) = 1$ if $x_k = x_i$, and 0 otherwise.

Direct application of the previous procedure gives an estimate of the marginal probabilities of all the variables when there is no evidence. If there is evidence, *i.e.*, some variables are instantiated, all samples that are not consistent with the evidence are discarded and the posterior probabilities are estimated from the remaining samples.

A disadvantage of logic sampling when evidence exists is that the data-efficiency is very low since many samples have to be discarded. This implies that a larger number of samples are required to have a relatively good estimation.

2.5.2 Likelihood weighting

Likelihood weighting generates samples in the same way as logic sampling, however when there is evidence the non-consistent samples are not discarded. Instead, each sample is given a weight according to the weight of the evidence for this sample. Given a sample of all non-instantiated nodes H and the evidence variables E, the weight of sample i is calculated as:

$$w_i = \mathbf{P}(E \mid H_i),$$

then the posterior probability of possible values of each variable is estimated as a weighted average over all n samples:

$$\mathbf{P}(X = x_k) = \frac{\sum_{i=1}^{n} w_i I_{x_k}(x_i)}{\sum_{i=1}^{n} w_i}.$$

3 Parameter learning

When the network structure is known, parameter learning of a Bayesian network consists in estimating the conditional probability tables from data. If we have sufficient and complete data for all the variables, the conditional probability table for each variable can be estimated based on the frequency of each value or combination of values via *maximum likelihood estimation* (MLE). For example, to estimate the conditional probability table of variable C with two parents A and B given the observed n samples, each entry of the table can be estimated according to

$$\mathbf{P}(C = c_k \mid A = a_i, B = b_j) = \frac{\sum_{i'=1}^{n} I_{a_i,b_j,c_k}(a_{i'}, b_{i'}, c_{i'})}{\sum_{i'=1}^{n} I_{a_i,b_j}(a_{i'}, b_{i'})}$$

with indicator function I. However, it can sometimes happen that we do not have a perfect dataset for learning the parameters.

3.1 Smoothing

When we estimate probabilities from data, it can sometimes happen that a particular event never occurs in the data set. This leads to the corresponding probability value being zero, implying an impossible case. Hence, if in the inference process this probability is considered, it will also make the result zero. This situation occurs, in many cases, because there is insufficient data to have a robust estimate of the parameters, and not because it is really an impossible event. This problem can be addressed by using some type of smoothing for the probabilities, eliminating zero probability values. From a Bayesian point of view, smoothing corresponding to estimate the posterior distribution of the parameters given some priors.

3.1.1 Uniform prior

One of the most common and simplest prior one can consider is a uniform distribution on the variable domain, which is called *Laplacian smoothing* (or *additive smoothing*). Consider a discrete variable X with m possible values. Given a dataset with n samples, the estimation of its probability will be the following:

$$\mathbf{P}(x_i) = \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n}, \quad i = 1, \ldots, m, \tag{3.1}$$

where I is the indication function of x_i and α is the smoothing parameter, with $\alpha = 0$ corresponding to no smoothing. When there is no observed sample, *i.e.*, $n = 0$, the estimated parameter distribution is simply the uniform prior

$$\mathbf{P}(x_i) = \frac{1}{m}, \quad i = 1, \ldots, m.$$

As the number of observed samples increases, the parameter estimation will converge to the true data distribution since in this case,

$$\lim_{n\to\infty} \mathbf{P}(x_i) = \lim_{n\to\infty} \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n} = \frac{\sum_{i'=1}^{n} I_{x_i}(x_{i'})}{n},$$

for all $i = 1, \ldots, m$.

3.1.2 Beta prior

Other than the non-informative uniform prior, we can try to integrate more expert information into the parameter estimation by considering an informative prior. For binary variables, we can build a prior based on the expectations of a beta distribution, $\text{Beta}(\alpha, \beta)$. The expected value of random variable $X \sim \text{Beta}(\alpha, \beta)$ is controlled by the *shape parameters* α and β, that is

$$\mathop{\mathbf{E}}_{\text{Beta}(\alpha,\beta)} X = \mathbf{P}(X = 1 \mid \alpha, \beta) = \frac{\alpha}{\alpha + \beta}.$$

Then similar to (3.1), given a dataset with n samples, the estimation of binary variable X can be obtained as

$$\mathbf{P}(X = 1) = \frac{\alpha + \sum_{i'=1}^{n} I_1(x_{i'})}{\alpha + \beta + n},$$

and $\mathbf{P}(X = 0) = 1 - \mathbf{P}(X = 1)$. In this case, the fraction $\frac{\alpha}{\alpha+\beta}$ determines the expert's prior for $X = 1$, while the denominator $\alpha + \beta$ defines the confidence about the prior. This means that a higher $\alpha + \beta$ value assigns more confidence on the prior while a lower value places more weight on the observations.

Example. *Smoothing with beta prior.* Assuming that an expert gives an estimate of $\mathbf{E}_{\text{Beta}(\alpha,\beta)} X = 0.7$ for a certain parameter, and that the experimental data provides 40 positive cases among 100 samples. The parameter estimation for different confidences assigned to the expert will be the following:

- Low confidence ($\alpha + \beta = 10$): $\mathbf{P}(X = 1) = \frac{7+40}{10+100} = 0.43$.
- Medium confidence ($\alpha + \beta = 100$): $\mathbf{P}(X = 1) = \frac{70+40}{100+100} = 0.55$.
- High confidence ($\alpha + \beta = 1000$): $\mathbf{P}(X = 1) = \frac{700+40}{1000+100} = 0.67$.

In the first case, the expert prior is dominated by the data, while in the third case the probability is closer to the expert's estimation, and the second case provides a compromise between them.

3.1.3 Dirichlet prior

The idea of the previous beta prior can be extended to general m-valued random variables by replacing the beta distribution with a Dirichlet distribution, $\mathrm{Dir}(\alpha)$, with the *concentration parameter* α being an m-vector. The expected value of each entry of an m-dimensional random vector $X \sim \mathrm{Dir}(\alpha)$ is

$$\mathop{\mathbf{E}}_{\mathrm{Dir}(\alpha)} X_i = \mathbf{P}(x_i \mid \alpha) = \frac{\alpha_i}{\mathbf{1}^T \alpha}, \quad i = 1, \ldots, m,$$

and we also have

$$\sum_{i=1}^{m} \mathop{\mathbf{E}}_{\mathrm{Dir}(\alpha)} X_i = 1.$$

We can view the m-dimensional random vector as a m-valued discrete random variable by assigning each of the entries an value. Hence, with the given n observations, we can estimate the probability distribution of X as

$$\mathbf{P}(x_i) = \frac{\alpha_i + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\mathbf{1}^T \alpha + n}, \quad i = 1, \ldots, m.$$

Similar to the beta prior, the fraction $\alpha_i / \mathbf{1}^T \alpha$ determines the expert's prior for $X = x_i$, and the confidence about the prior is controlled by the value of $\mathbf{1}^T \alpha$.

3.2 Missing data

Another common situation that we may have during parameter learning is to have incomplete data. There are two basic cases:

- Missing values: in some samples there are missing values for one or more variables.
- Hidden nodes: a variable or set of variables in the model cannot be observed.

For dealing with missing values, there are several alternatives. One trivial approach would be to remove all the samples with missing values. This would be acceptable only if there is sufficient data. Another alternative without removing any samples from the dataset is to substitute the missing value by the most common value of that variable based on all available observations. However, this may

bias the model since the information from the other variables is not taken into account. In general the best option would be to estimate the missing value based on the values of the other variables in the corresponding sample. In this case, we first learn the parameters of the Bayesian network based on the samples with complete observations, and then for each sample with missing values applying the following process:

1. Instantiate all the known variables in the sample.
2. Through probabilistic inference obtain the posterior probabilities of the missing variables.
3. Assign to each unknown variable the value with highest posterior probability, or sample one value according to the posterior probability.
4. Add this completed sample to the database.

Finally we can re-estimate the model parameters based on the completed dataset.

For hidden nodes, the approach to estimate their parameters is based on the expectation-maximization (EM) algorithm. The algorithm starts by initializing the missing parameters with random values. Then for each EM-iteration, in the E-step, the missing data values are estimated based on the current parameters; in the M-step, the parameters are updated based on the estimated data. This will be repeated multiple times until the parameters get converge.

3.3 Discretization

Usually Bayesian networks consider discrete or categorical variables. Although there are some developments for continuous variables, these are restricted to certain distributions, in particular Gaussian variables and linear relations. An alternative to include continuous variables in Bayesian networks is to discretize them, *i.e.*, transform them to categorical variables. Discretization methods can be unsupervised and supervised.

3.3.1 Unsupervised discretization

Unsupervised discretization do not consider the task for which the model is going to be used, meaning that the discretization intervals for each variable are determined independently. The two main types of unsupervised discretization approaches are *equal width* and *equal data*. Equal width consists in dividing the range of a variable into k equal bins, such that each bin has a size of $(\sup X - \inf X)/k$. The number of intervals k is usually assigned by the user. Equal data divides the range of the variable into k intervals, such that each interval includes the same number of data points from the training dataset. In other words, if there are n samples, each interval will contain n/k data points, but the intervals will not necessarily have the same width.

3.3.2 Supervised discretization

Supervised discretization incorporates the task to be performed with the model, such that the variables are discretized to optimize this task, for instance classification accuracy. This can be posed as an optimization problem. For example, consider a continuous feature variable X and a categorical class variable C. Given n training samples with each one having a value for C and X, the problem is to determine the optimal partition of $(\inf X, \sup X)$ such that the classification accuracy is maximized. This is then a combinatorial optimization problem that is computationally complex. Different search approaches can be used, including basic ones such as *hill-climbing* or more sophisticated methods like *simulated annealing* and *genetic algorithms*.

4 Structure learning

4.1 Tree learning

For a particular case where the dependencies between random variables can be represented with a tree-structure, the structure learning procedure can be separated into two steps: (1) learning the skeleton of the tree, *i.e.*, establishing undirected edges between variables, and (2) determining the direction of the edges.

4.1.1 Skeleton learning

The *Chow-Liu procedure* (CLP) obtains the skeleton of a tree, but does not provide the directions of the arcs. Given a set of n random variables $X = \{X_1, \ldots, X_n\}$, we would like to find the tree structure that best approximate the joint distribution of these variables $\mathbf{P}(x)$. Let $\widetilde{\mathbf{P}}(x)$ be the approximated joint distribution obtained from some tree including these variables, the skeleton learning problem consists in minimizing the distance between distributions $\mathbf{P}(x)$ and $\widetilde{\mathbf{P}}(x)$, according to the *KL-divergence* measure:

$$D_{\mathrm{KL}}(\mathbf{P}, \widetilde{\mathbf{P}}) = \sum_{x \in X} \mathbf{P}(x) \log \left(\frac{\mathbf{P}(x)}{\widetilde{\mathbf{P}}(x)} \right). \tag{4.1}$$

However, evaluating the KL-divergence for all possible trees is very expensive. As an alternative of the objective (4.1), let

$$I(X_i, X_j) = \sum_{x_i \in X_i, x_j \in X_j} \mathbf{P}(x_i, x_j) \log \left(\frac{\mathbf{P}(x_i, x_j)}{\mathbf{P}(x_i)\,\mathbf{P}(x_j)} \right)$$

be the *mutual information* between any pair of variables $X_i \in X$, $X_j \in X$, we define the weight $W(X)$ as the sum of the mutual information of the edges that constitute the tree $G = (X, E)$ as

$$W(X) = \sum_{(X_i, X_j) \in E} I(X_i, X_j) = \sum_{i=1}^{n-1} I(X_i, \mathbf{pa}(X_i)). \tag{4.2}$$

(Here we assume the root note is indexed as X_n in the tree without loosing of generality.) It can be shown that minimizing (4.1) is equivalent to maximizing (4.2) over the set of edges E. Therefore, obtaining the optimal tree is equivalent to finding the *maximum weight spanning tree*, according to the following procedure:

1. Obtain the mutual information $I(X_i, X_j)$ for all pairs of variables $X_i \in X$, $X_j \in X$.
2. Order the mutual information values in descending order.
3. Select the pair (X_i, X_j) with maximum $I(X_i, X_j)$ and connect the two variables with an edge, this constitutes the initial tree.
4. Add the pair with the next highest mutual information to the tree if they do not make a cycle, otherwise skip it and continue with the following pair.
5. Repeat the previous step until all the variables are in the tree.

4.1.2 Direction learning

Based on the learnt skeleton of the tree from CLP, one trivial way of assigning edge directions would be randomly select one node as the tree root and assign directions to the edges starting from this root. Another option is to obtain directions using external semantics, or using higher order dependency tests. We will introduce the last alternative subsequently.

As discussed in §1.1.2, given three variables X, Y, and Z, there are three possibilities for their dependency:

- Sequential: $X \rightarrow Y \rightarrow Z$.
- Divergent: $X \leftarrow Y \rightarrow Z$.
- Convergent: $X \rightarrow Y \leftarrow Z$.

The first two cases are indistinguishable under statistical independence testing, since in both cases X and Z are independent given Y. However the third case is different, where X and Z are not independent given Y. Hence, this case can be used to determine the directions of the two arcs that connect these three variables, and once we have identified a convergent structure, we can apply this knowledge to learn the directions of other arcs using independence tests. With this, the following algorithm can be used for learning the direction of a tree skeleton:

1. Iterate over the network until a convergent variable triplet is found. We will call the variable to which the arcs converge a *multi-parent node*.
2. Starting with a multi-parent node, determine the directions of other arcs using independence tests for variable triplets. Continue this procedure until it is no longer possible.
3. Repeat the first two steps until no other directions can be determined.

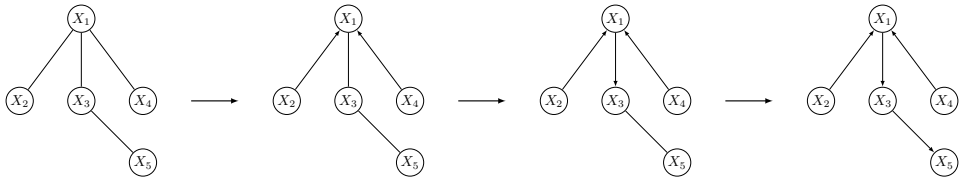


Figure 13 An example of learning the edge directions given a tree skeleton using independence tests.

However, we should note that there is no guarantee that the direction for all the arcs in the tree can be obtained via this procedure. If any arcs are left undirected when the algorithm quits, external semantics can be used to infer their directions.

Example. *Direction learning using independence tests.* Given a tree skeleton shown in figure 13, we first perform statistical independence tests for the variable triplet $\{X_1, X_2, X_4\}$. Suppose we find that X_2 are dependent of X_4 given X_1, meaning the variable triplet $\{X_1, X_2, X_4\}$ falls into the convergent substructure. Hence, we can assign the direction for edge (X_1, X_2) and (X_1, X_4) as $X_2 \rightarrow X_1 \leftarrow X_4$. Based on this knowledge, we can further test the independence relationship between triplet $\{X_1, X_2, X_3\}$ and $\{X_1, X_3, X_4\}$. If we have $(X_2 \perp\!\!\!\perp X_3 \mid X_1)$ and $(X_3 \perp\!\!\!\perp X_4 \mid X_1)$, the direction of edge (X_1, X_3) must be $X_1 \rightarrow X_3$. Otherwise, both the variable triplets $\{X_1, X_2, X_3\}$ and $\{X_1, X_3, X_4\}$ fall into convergent substructure and we can thus assign the edge direction $X_1 \leftarrow X_3$. Finally, the same process can be applied for triplet $\{X_1, X_3, X_5\}$ to determine the direction of edge (X_3, X_5).

4.2 Score-based methods

Score-based methods view the structure learning problem as a combinatorial optimization problem with respect to some type of scoring function over the network structure. Since this problem is generally NP-hard, we resort to heuristic search techniques, such as hill-climbing, simulated annealing, genetic algorithms, etc. As can be expected, one of the most important decisions we must make in this framework is the choice of scoring function S.

4.2.1 Likelihood score

The likelihood score is defined as the probability of observing the dataset $\mathcal{D}$ given the graph G and its parameters θ_G, typically expressed in logarithmic form:

$$S_{\text{LL}}(G) = l_{\mathcal{D}}(\theta_G) = \log \mathbf{P}(\mathcal{D} \mid \theta_G, G). \tag{4.3}$$

The network structure found by directly maximizing the log-likelihood score (MLE estimation) can be highly complex, which usually implies overfitting the data (poor generalization) and also makes inference more complex.

4.2.2 Bayesian score

An alternative metric which avoids overfitting is expressed by following a Bayesian approach, obtaining the posterior probability of the structure given the data with the Bayes rule:

$$\mathbf{P}(G \mid \mathcal{D}) = \frac{\mathbf{P}(\mathcal{D} \mid G)\,\mathbf{P}(G)}{\mathbf{P}(\mathcal{D})}.$$

Since the denominator $\mathbf{P}(\mathcal{D})$ is a constant that does not depend on the structure, it can be discarded from the metric. Thus we define the Bayesian score as:

$$S_{\mathrm{B}}(G) = \log \mathbf{P}(\mathcal{D} \mid G) + \log \mathbf{P}(G),$$

which is again, for convenience, expressed in logarithmic form. The probability $\mathbf{P}(G)$ is the prior over network structures, allowing us to prefer some structures over others. The likelihood term $\mathbf{P}(\mathcal{D} \mid G)$ is called the marginal likelihood of the data since its obtained by marginalizing out the unknown model parameters θ_G:

$$\mathbf{P}(\mathcal{D} \mid G) = \int_{\theta_G} \mathbf{P}(\mathcal{D} \mid \theta_G, G)\,\mathbf{P}(\theta_G \mid G)\; d\theta_G, \tag{4.4}$$

where $\mathbf{P}(\mathcal{D} \mid \theta_G, G)$ is the likelihood of the data given the network G and its parameter θ_G, and $\mathbf{P}(\theta_G \mid G)$ is the prior distribution over different parameter values for the network G.

It is important to realize that maximizing the marginal likelihood (4.4) is quite different from maximizing the likelihood score (4.3). Both terms examine the likelihood of the data given the structure. The maximum likelihood score evaluates the likelihood of the training data using the best parameter values for the given dataset. This estimate is realistic only if these parameters are also reflective of the data in general, a situation that never occurs. In contrast, in the Bayesian approach, by integrating $\mathbf{P}(\mathcal{D} \mid \theta_G, G)$ over the different choices of parameters θ_G, we are measuring the expected likelihood, averaged over different possible choices of θ_G. Thus, we are being more conservative in our estimate of the goodness of the model, which contributes to avoid overfitting.

In practice, the value of the marginal likelihood $\mathbf{P}(\mathcal{D} \mid G)$ depends on the parameter prior $\mathbf{P}(\theta_G \mid G)$ that we select, as well as the number of samples in the dataset, etc. For example, if we use a Dirichlet parameter prior for all parameters in the network, then, when the number of samples in the dataset $n \to \infty$, we have the *Bayesian information criterion* (BIC):

$$\begin{aligned} S_{\mathrm{BIC}}(G) &= l_{\mathcal{D}}(\theta_G) - \frac{k}{2}\log n \\ &= \log \mathbf{P}(\mathcal{D} \mid \theta_G, G) - \frac{k}{2}\log n, \end{aligned} \tag{4.5}$$

where k is the number of parameters in the model and n is the number of samples in the dataset. From this example, we can see that the Bayesian score seems to be biased toward simpler structures, but as it gets more data, it is willing to recognize that a more complex structure is necessary. In other words, it appears to trade off fit to data with model complexity, thereby reducing the extent of overfitting.

4.3 PC algorithm

The *PC algorithm* first recovers the skeleton (underlying undirected graph) of the Bayesian network, and then it determines the direction of the edges. To determine the skeleton, it starts from a fully connected undirected graph, and determines the conditional independence of each pair of variables given some subset of the other variables. For this it assumes that there is a procedure that can determine if two variables, X and Y, are independent given a subset of variables Z. For example, using statistical tests or by calculating the *conditional cross-entropy measure*. If the independence measure is below some threshold value set according to a certain confidence level, the edge between the pair of variables is eliminated. These tests are iterated for all pairs of variables in the graph. Then in the second step, where the direction of the edges are determined, the same independence testing procedure based on variable triplets as introduced in §4.1.2 can be applied.

If the set of independencies are *faithful* to a graph, meaning that that conditional independence relations are due to the causal structure rather than because of accidents in parameter values, and the independence tests are perfect, the algorithm produces a graph equivalent to the original one.

Bibliography

An introduction to Bayesian networks is given in the classic book by Pearl [Pea88]. The books by Jensen and Nielsen [JN07] and by Neapolitan [Nea90] also includes some useful information on Bayesian networks. A more recent account with emphasis on modeling, inference, and complexity analysis is given in [Dar09]. Other books with more emphasis on applications are [PNM08] and [KN10].

A formal definition about d-separation, including some useful properties and corresponding mathematical analysis can be found in [Pea09, §1.2].

An overview of canonical models is presented in the article [DD06].

The belief propagation algorithm was initially proposed in [Pea86] for the inference of singly connected Bayesian networks. This idea can be generalized to multi-connected networks, which is called the *loopy propagation* algorithm. It has been found empirically that for certain structures this algorithm converges to the true posterior probabilities, but for other structures that it does not converge, it can only provide approximate solution of the inference problem [MWJ13].

The computational complexity in terms of space and time of the variable elimination algorithm is determined by the size of the factors. The book [Dar09] can be referred to for some detailed information.

The book [Pea88] provides a detailed introduction about conditioning algorithms. Several variants of the conditioning algorithm have been proposed, including local conditioning [Díe96] and recursive conditioning [Dar01].

In this chapter we do not include specific algorithms for finding the minimal triangulations of graphs. The readers can refer to the survey by Heggernes [Heg06] for more information.

The junction tree algorithm was initially introduced in [LS88]. There are two main variations on the junction tree algorithm, which are known as the Hugin [JA13] and Shafer-Shenoy [SS90] architectures. The description in this chapter is based on the Hugin architecture. The main differences between them are in the information they store, and in the way they compute the messages. These differences have implications in their computational complexity. In general, the Shafer-Shenoy architecture will require less space but more time.

In this chapter we introduced inference methods for discrete-valued Bayesian networks. When dealing with continuous variables, probabilistic inference techniques have been developed for some distribution families, in particular Gaussian variables [Pea88].

For the equal width discretization method, specially for Bayesian classifiers, one useful way to determine the number of intervals is called *proportional k-interval discretization* (PKID) [YW01]. This strategy seeks a trade-off between the bias and variance of the parameter estimates by adjusting the number and size of intervals to the number of training instances. Given a continuous variable with n training instances, it is discretize to $\sqrt{n}$ intervals, with $\sqrt{n}$ instances in each interval. This method was compared empirically with other discretization methods for learning naive Bayesian classifiers, where PKID achieves the lowest mean error.

For heuristic methods solving combinatorial optimization problems, the books [PS98] and [RN16] provide a in depth introduction about the hill-climbing algorithm and its variants. Information about simulated annealing method can be found in [KGV83]. The

paper from Holland [Hol73] can be referred to for more details about the class of genetic algorithms.

The Chow-Liu procedure was first described in the paper [CL68]. Pearl [Pea88] provides a modern analysis of the algorithm as a Bayesian network. The statistical independence test based algorithm for assigning directions given a tree skeleton was initially introduced in [RP87].

Apart from the Bayesian information criterion that we introduced in this chapter, there are many other Bayesian scores based on different parameter priors and assumptions that have been widely used for structure learning, including the Bayesian-Dirichlet (BD) score [HGC95], BDe score ('e' for likelihood-equivalence) [HGC95], BDeu score ('u' for uniform joint distribution [Bun91]), and K2 score [CH92].

For a detailed discussion about the PC algorithm, and the *faithfulness condition* of a set of independencies to a graph, one can referred to [SGS01].

References

[Bun91] W. Buntine. Theory refinement on Bayesian networks. In *Uncertainty Proceedings 1991*, pages 52–60. Elsevier, 1991.

[CH92] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[Dar01] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[Dar09] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[DD06] F. J. Dıez and M. J. Druzdzel. Canonical probabilistic models for knowledge engineering. Technical Report CISIAD-06, UNED, 2006.

[Díe96] F. J. Díez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87(1-2):1–20, 1996.

[Heg06] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[HGC95] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[Hol73] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[JA13] F. Jensen and S. K. Anderson. Approximations in Bayesian belief universe for knowledge based systems. *arXiv*, 1304.1101, 2013.

[JN07] F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007.

[KGV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KN10] K. B. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence*. CRC Press, 2010.

[LS88] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[MWJ13] K. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. *arXiv*, 1301.6725, 2013.

[Nea90] R. E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley & Sons, 1990.

[Pea86] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[Pea09] J. Pearl. *Causality*. Cambridge University Press, 2nd edition, 2009.

[PNM08] O. Pourret, P. Naim, and B. Marcot. *Bayesian Networks: A Practical Guide to Applications*. John Wiley & Sons, 2008.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

[RN16] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2016.

[RP87] G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In *Proceedings of the 3rd Conference on Uncertainty in Artificial Intelligence*, pages 222–228, 1987.

[SGS01] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2001.

[SS90] P. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In *Machine Intelligence and Pattern Recognition*, volume 9, pages 169–198. Elsevier, 1990.

[YW01] Y. Yang and G. I. Webb. Proportional k-interval discretization for naive-Bayes classifiers. In *Machine Learning: ECML 2001*, pages 564–575. Springer, 2001.