7

Learning the Structure of Bayesian Networks

Consider the following situation. Some agent produces samples of cases from a Bayesian network N over the universe $\mathcal{U}$. The cases are handed over to you, and you are asked to reconstruct the Bayesian network from the cases. This is the general setting for structural learning of Bayesian networks. In the real world you cannot be sure that the cases are actually sampled from a "true" network, but this we will assume. We will also assume that the sample is fair. That is, the set $\mathcal{D}$ of cases reflects the distribution $P_N(\mathcal{U})$ determined by N. In other words, the distribution $P^{\#}_{\mathcal{D}}(\mathcal{U})$ of the cases is very close to $P_N(\mathcal{U})$. Furthermore, we assume that all links in N are *essential*, i.e., if you remove a link, then the resulting network cannot represent $P(\mathcal{U})$. Mathematically, it can be expressed as follows: if $\mathrm{pa}(A)$ are the parents of A, and B is any of them, then there are two states b_1 and b_2 of B and a configuration $\mathbf{c}$ of the other parents such that $P(A|b_1, \mathbf{c}) \neq P(A|b_2, \mathbf{c})$.

The task is now to find a Bayesian network, M, close to N. In principle this can be done by performing parameter learning for all possible structures, and then selecting as candidates those models for which $P_M(\mathcal{U})$ is close to $P^{\#}_{\mathcal{D}}(\mathcal{U})$. However, by following this very simple approach we are faced with three problems, which are fundamental for learning Bayesian networks. First of all, the space of all Bayesian network structures is extremely large. In fact, it has been shown that the number of different structures, $f(n)$, grows more than exponentially in the number n of nodes (some example calculations can be found in Table 7.1):

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} \frac{n!}{(n-i)!n!} 2^{i(n-i)} f(n-1). \tag{7.1}$$

Secondly, when searching through the network structures, we may end up with several equally good candidate structures. Since a Bayesian network over a complete graph can represent any distribution over its universe, we know that we will always have several candidates, but a Bayesian network

Nodes	Number of DAGs	Nodes	Number of DAGs
1	1	13	$1.9 \cdot 10^{31}$
2	3	14	$1.4 \cdot 10^{36}$
3	25	15	$2.4 \cdot 10^{41}$
4	543	16	$8.4 \cdot 10^{46}$
5	29281	17	$6.3 \cdot 10^{52}$
6	$3.8 \cdot 10^{6}$	18	$9.9 \cdot 10^{58}$
7	$1.1 \cdot 10^{9}$	19	$3.3 \cdot 10^{65}$
8	$7.8 \cdot 10^{11}$	20	$2.35 \cdot 10^{72}$
9	$1.2 \cdot 10^{15}$	21	$3.5 \cdot 10^{79}$
10	$4.2 \cdot 10^{18}$	22	$1.1 \cdot 10^{87}$
11	$3.2 \cdot 10^{22}$	23	$7.0 \cdot 10^{94}$
12	$5.2 \cdot 10^{26}$	24	$9.4 \cdot 10^{102}$

Table 7.1. The table shows the number of different DAGs that can be generated for a given number of nodes. For example, there exist $4.2 \cdot 10^{18}$ different DAGs with 10 nodes.

over a complete graph will hardly be the correct answer. If so, it is a very disappointing answer.

Thirdly, we have the problem of *overfitting*: the selected model is so close to $P_{\mathcal{D}}^{\#}(\mathcal{U})$ that it also covers the smallest deviances from $P_N(\mathcal{U})$. Again, a complete graph can represent $P_{\mathcal{D}}^{\#}(\mathcal{U})$ exactly, but $\mathcal{D}$ may have been sampled from a sparse network.

There are basically two methods used for learning the structure of Bayesian networks; *constraint-based* and *score-based*. The constraint based methods establish a set of conditional independence statements holding for the data, and use this set to build a network with d-separation properties corresponding to the conditional independence properties determined. The score-based methods produce a series of candidate Bayesian networks, calculate a score for each candidate, and return a candidate of highest score.

To emphasize the focus on structural learning we shall use the following convention: A Bayesian network $M = (S, \boldsymbol{\theta}_S)$ consists of a network structure, S, and a set of parameters, $\boldsymbol{\theta}_S$, where the parameters determine the conditional probabilities of the model. The structure S consists of an acyclic directed graph, $G = (\mathcal{U}, \mathcal{E})$, together with a specification of the state space for each node/variable in the graph.

7.1 Constraint-Based Learning Methods

We shall first consider the following problem: we have to determine the structure of a Bayesian network, and the only source of information is an oracle that correctly answers queries of the type, "is the variable A d-separated from the variable B given the set $\mathcal{X}$?", later we shall replace the oracle with a database

for answering the queries. We let $I(A, B, \mathcal{X})$ denote that A is d-separated from B given $\mathcal{X}$. We use $I(A, B)$ as shorthand for $I(A, B, \emptyset)$, and if $\mathcal{X}$ consists of only one element C, we write $I(A, B, C)$.

The method consists in first determining the skeleton of the network, and afterward directing the links.

Definition 7.1. *The* skeleton *of a Bayesian network N is the undirected graph obtained by removing directions from all arcs in N.*

The skeleton can quite easily be established through a series of questions to the oracle: if there is a link between A and B, then they cannot be d-separated. That is, the link $A - B$ is part of the skeleton if and only if $\neg I(A, B, \mathcal{X})$ for all $\mathcal{X}$ not containing A or B. As a starting point, let us assume that we have the skeleton.

7.1.1 From Skeleton to DAG

Consider the skeleton in Figure 7.1(a), and assume that the only conditional independence found is $I(A, B)$. This means that A and B are not independent given C, and therefore Figure 7.1(b) is the only possible directed graph with d-separation properties corresponding to the conditional independences found.

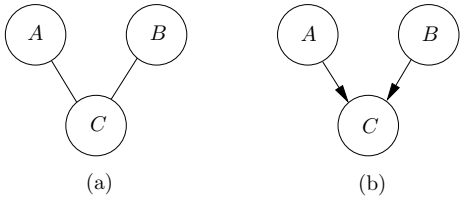


Fig. 7.1. (a) A skeleton for the set $\{I(A, B)\}$. (b) the corresponding DAG.

This observation can be generalized as illustrated in Figure 7.2, where C must be a child of A and B if $I(A, B)$ or $I(A, B, D)$.

Rule 1 [introduction of v-structures]: If you have three nodes, A, B, C, such that $A - C$ and $B - C$, but not $A - B$, then introduce the v-structure $A \rightarrow C \leftarrow B$ if there exists an $\mathcal{X}$ (possibly empty) such that $I(A, B, \mathcal{X})$ and $C \notin \mathcal{X}$.

As an example, consider the skeleton in Figure 7.3(a) with independences $I(A, B)$, $I(B, C)$, $I(A, B, C)$, $I(B, C, A)$, $I(C, D, A)$, $I(B, C, \{D, A\})$, $I(C, D, \{A, B\})$, $I(B, E, \{C, D\})$, $I(A, E, \{C, D\})$, $I(B, C, \{A, D, E\})$, $I(A, E, \{B, C, D\})$, $I(B, E, \{A, C, D\})$. Consider the chain $C - E - D$. Since E is not a member of any conditioning set yielding C and D independent, we introduce the v-structure $C \rightarrow E \leftarrow D$. In the same way we introduce the v-structure

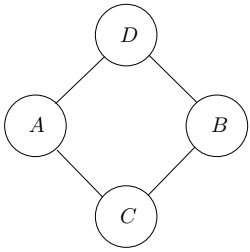


Fig. 7.2. $\{A, C, B\}$ are connected in an undirected chain, but there is another path between A and B. If also $I(A, B)$ or $I(A, B, D)$, then C must be a child of A and B.

$A \rightarrow D \leftarrow B$ (see Figure 7.3(b)) With these two v-structures, there cannot be more of them. This is also confirmed by the conditional independences, and since they give no clue as to the remaining link, $A - C$, it may be oriented in any direction (see Figure 7.3(c)).

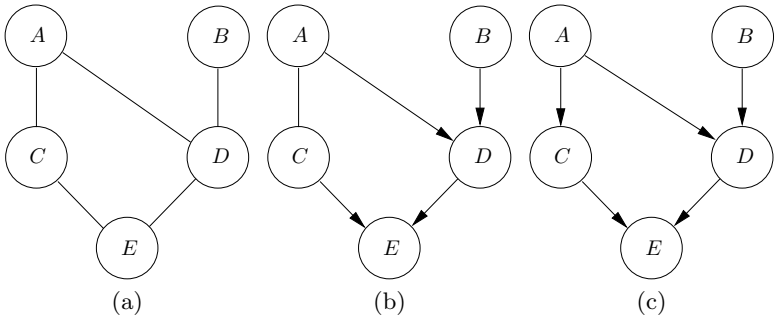


Fig. 7.3. (a) A skeleton. (b) Two v-structures introduced through rule 1. (c) A full DAG

After the v-structures have been introduced, we give a direction to the remaining links using the following rules:

Rule 2 [Avoid new v-structures]: When Rule 1 has been exhausted, and you have $A \rightarrow C - B$ (and no link between A and B), then direct $C \rightarrow B$.

Rule 3 [Avoid cycles]: If $A \rightarrow B$ introduces a directed cycle in the graph, then do $A \leftarrow B$.

Rule 4 [Choose randomly]: If none of the rules 1–3 can be applied anywhere in the graph, choose an undirected link and give it an arbitrary direction.

For example, having found the v-structures in Figure 7.3 (b), we can choose any direction for $A - C$ (Figure 7.3 (c)).

Example 7.1. Consider the graph in Figure 7.4(a). The only v-structure found is $C \rightarrow F \leftarrow D$. Rule 2 yields the direction $F \rightarrow G$ (Figure 7.4(b)). None of the Rules 1–3 can be applied, and we choose the direction $D \leftarrow E$ (Figure 7.4(c)). Now Rule 2 yields $D \rightarrow A$ and $D \rightarrow B$ (Figure 7.4(d)), and in turn, Rule 2 yields $A \rightarrow C$ (Figure 7.4(e)). Now none of the Rules 1–3 can be applied. We use rule 4 and choose $A \rightarrow B$ (Figure 7.4(f)).

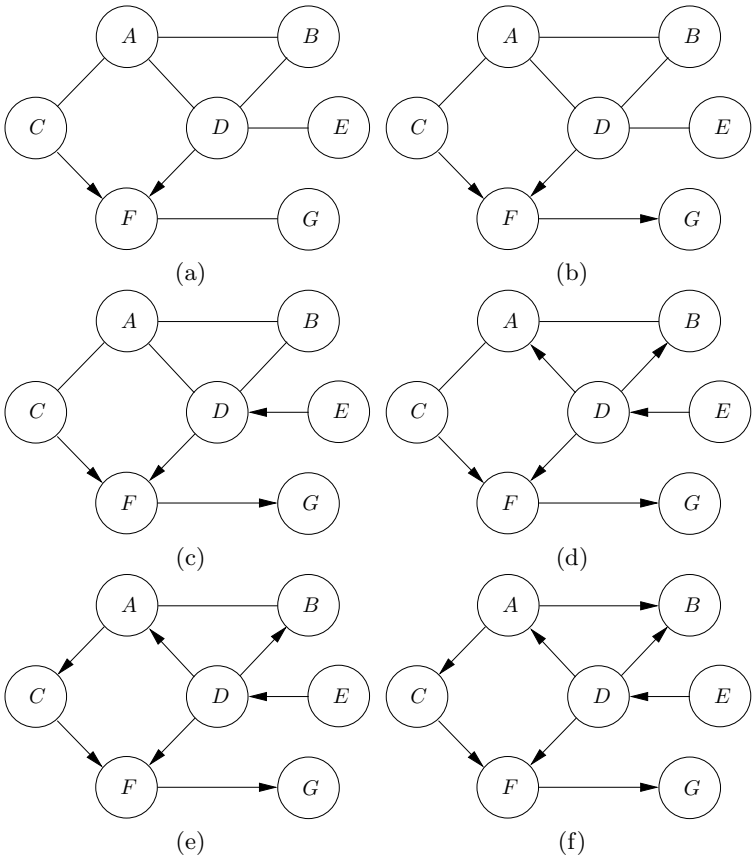


Fig. 7.4. A sequence of applications of Rules 2 and 4.

The application of Rules 2–4 raises various questions. For example, Rule 4 opens up for several solutions. If in the example above we had chosen $D \rightarrow E$ rather than $D \leftarrow E$, the solution could have been the DAG in Figure 7.5.

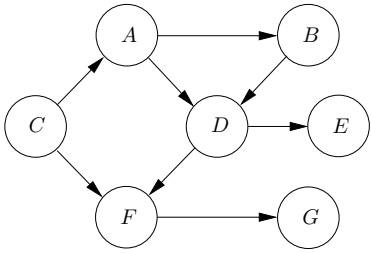


Fig. 7.5. A resulting DAG if we had chosen $D \rightarrow E$ rather than $D \leftarrow E$.

A solution represents a family of probability distributions over the universe $\mathcal{U}$; a distribution for each setting of the parameters. From a statistical point of view, the various solutions are equivalent; they have the same d-separation properties. Equivalently, they represent the same family of distributions. Furthermore, a maximal likelihood setting of the parameters in one graph will have a corresponding parameter setting in any other graph, and this parameter setting is also of maximal likelihood.

A more fundamental problem is whether there in fact is a solution. If the oracle is reliable, then the skeleton and the v-structures are correct, and therefore there must be a way to direct the remaining links so that the generative model is established. Moreover, any other solution will also be valid (see Section 7.3.2). Finally, you might fear that the choice in Rule 4 may lead us into a blind alley. However, it has been proven that this will not happen.

7.1.2 From Independence Tests to Skeleton

Since consulting the oracle has a price, we wish to reduce the number of questions. We use the answers from the oracle when establishing the skeleton and when introducing v-structures. The following theorem helps to reduce the number of questions.

Theorem 7.1. *The nodes A and B are not linked in N if and only if $I(A, B, \mathrm{pa}(A))$ or $I(A, B, \mathrm{pa}(B))$.*

Proof. Clearly, if $I(A, B, \mathcal{X})$ for any $\mathcal{X}$, then A and B are not linked.

Assume now that A and B are not linked in N, and construct the ancestral graph for $\{A, B\}$ (see Section 2.2.1). If there is a path in this graph from B to A not passing through $\mathrm{pa}(B)$, then B is an ancestor of A, and all paths from B to A must pass through $\mathrm{pa}(A)$.

□

The theorem ensures that it is sufficient to ask questions of the form $I(A, B, \mathcal{X})$, where $\mathcal{X}$ is a subset of A's or B's neighbors. It is used in the *PC algorithm* to focus on local independence questions.

Algorithm 7.1 [The PC algorithm: test sequence]

1. *Start with the complete graph;*
2. $i := 0$*;*
3. **while** *a node has at least* $i + 1$ *neighbors*
   - **for all** *nodes* A *with at least* $i + 1$ *neighbors*
     - **for all** *neighbors* B *of* A
       - **for all** *neighbor sets* $\mathcal{X}$ *such that* $|\mathcal{X}| = i$ *and* $\mathcal{X} \subseteq (\mathrm{nb}(A) \backslash \{B\})$
         - **if** $I(A, B, \mathcal{X})$ **then** *remove the link* $A - B$ *and store* "$I(A, B, \mathcal{X})$"
   - $i := i + 1$

□

7.1.3 Example

Assume that the cases are a faithful sample of the Bayesian network in Figure 7.6(a). We start with the complete graph in Figure 7.6(b) and ask the questions $I(A, B)?$, $I(A, C)?$, $I(A, D)?$, $I(A, E)?$, $I(B, C)?$, $I(B, D)?$, $I(B, E)?$, $I(C, D)?$, $I(C, E)?$, $I(D, E)?$.

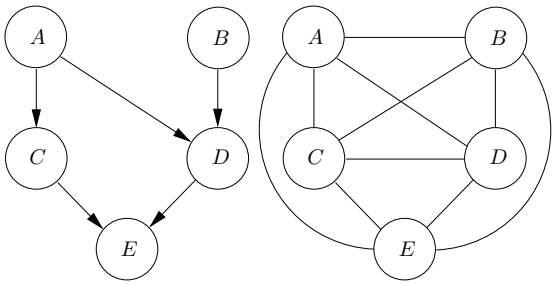


Fig. 7.6. (a) The Bayesian network from which the cases have been sampled.(b) The starting graph for the PC algorithm.

We get "yes" for $I(A, B)?$ and $I(B, C)?$; the links $A - B$ and $B - C$ are removed (see Figure 7.7(a)), and i is set to 1.

We now ask $I(A, C, E)?$, $I(B, C, D)?$, $I(B, C, E)?$, $I(B, D, C)?$, $I(B, D, E)?$, $I(B, E, C)?$, $I(B, E, D)?$, $I(C, B, A)?$, $I(C, D, B)?$, $I(C, D, A)?$. The last question has the answer "yes"; we remove the link $C - D$ and continue; $I(C, E, A)?$, $I(C, E, B)?$, $I(D, B, E)?$, $I(D, E, B)?$, $I(E, A, B)?$, $I(E, A, D)?$, $I(E, B, A)?$, $I(E, C, B)?$, $I(E, C, D)?$, $I(E, D, A)?$, $I(E, D, C)?$.

Next, for $i = 2$ we ask questions like $I(A, C, \{D, E\})?$, and we get affirmative answers for $I(B, E, \{C, D\})?$ and $I(A, E, \{C, D\})?$. The result is shown

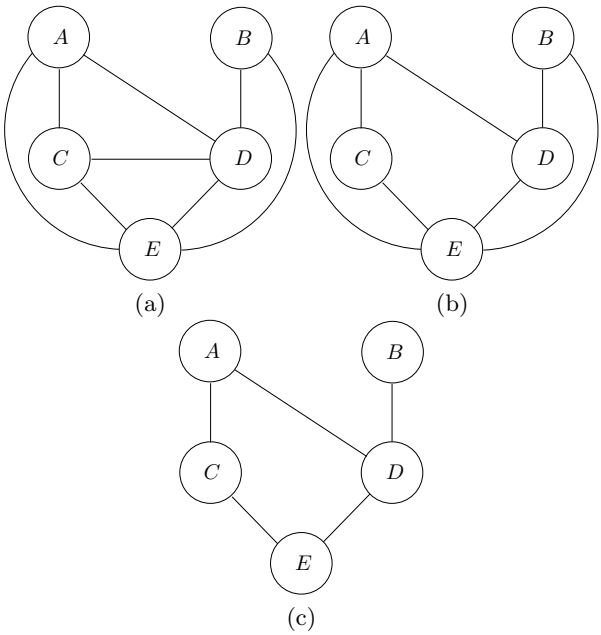


Fig. 7.7. (a) The result after testing all unconditional independences ($i = 0$). (b) After testing with a single conditioning variable. (c) After testing with two conditioning variables.

in Figure 7.7(c). Setting $i = 3$ we realize that no node has four neighbors, and the algorithm terminates.

To sum up, we get the skeleton in Figure 7.7(c) together with the conditional independences $I(A, B)$, $I(B, C)$, $I(C, D, A)$, $I(A, E, \{C, D\})$, and $I(B, E, \{C, D\})$. They are sufficient for applying Rules 1–4.

The PC-algorithm has the following property, which is easily seen from the construction and Theorem 7.1.

Property 1: If the case set is a faithful sample from a Bayesian network, N, then the graph resulting from the PC-algorithm is the skeleton of N.

We also have the following property, which allows us to establish the direction of the arcs.

Property 2: The conditional independences found by the PC-algorithm are sufficient for determining the v-structures.

Let namely $A - C - B$ be a chain, and assume that the PC-algorithm found $I(A, B, \mathcal{X})$. We know that the two links are part of the skeleton, and if $C \notin \mathcal{X}$ then the only way to direct the links will be to introduce the v-structure $A \rightarrow C \leftarrow B$. On the other hand, if $C \in \mathcal{X}$ we cannot have a v-structure.

The Necessary Path Condition

The number of queries to the oracle can be further reduced. Consider the situation in Figure 7.8, where the links $A - D$ and $A - C$ have been removed. Then we need not ask for $I(A, B, D)$ (or $I(A, B, C)$), since no path between A and B passes D (similar for C). This is called the *necessary path condition*: only ask $I(A, B, \mathcal{X})$ for sets $\mathcal{X}$, where all members of $\mathcal{X}$ occur on a path between A and B.

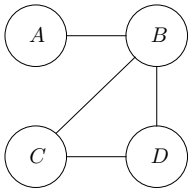


Fig. 7.8. D cannot block any path between A and B.

7.1.4 Constraint-Based Learning on Data Sets

When learning structure, you do not have an oracle for queries of the type $I(A, B, \mathcal{X})$. Instead, you have a data set $\mathcal{D}$, which you may analyze for conditional independences. We shall use the notation $I_{\mathcal{D}}(A, B, \mathcal{X})$ for conditional independence in the distribution determined by $\mathcal{D}$. We shall assume that $\mathcal{D}$ is sampled from a Bayesian network N.

Definition 7.2. *$\mathcal{D}$ is a* faithful sample *from N if the following holds: A and B are d-separated in N given $\mathcal{X}$ if and only if $I_{\mathcal{D}}(A, B, \mathcal{X})$.*

If $\mathcal{D}$ is faithful to N, we can use a test for independence in $\mathcal{D}$ as oracle. For this, *conditional mutual information* can be used.

$$\mathrm{CMI}(A, B|\mathcal{X}) = \sum_{\mathcal{X}} P^{\#}(\mathcal{X}) \sum_{A,B} P^{\#}(A, B|\mathcal{X}) \log_2 \frac{P^{\#}(A, B|\mathcal{X})}{P^{\#}(A|\mathcal{X})P^{\#}(B|\mathcal{X})}. \quad (7.2)$$

It holds (Exercise 7.5) that

$$I_{\mathcal{D}}(A, B, \mathcal{X}) \Leftrightarrow \mathrm{CMI}(A, B|\mathcal{X}) = 0.$$

Based on the data set, the oracle will calculate an estimate of $\mathrm{CMI}(A, B|\mathcal{X})$, and then it performs a χ^2-test on the hypothesis $\mathrm{CMI}(A, B|\mathcal{X}) = 0$, and the user decides on a significance level. A high significance level means that fewer

links are removed. Because any test has false positives as well as false negatives, there is a risk that links that should have been removed are not removed, and vice versa. The error rate is closely related to the sample size. The smaller the sample size, the more independences will be accepted and the fewer links inserted.

In a real-world learning situation, you may, for example, get into the situation illustrated in Figure 7.9: you have $\neg I(A, B), \neg I(A, C), \neg I(B, C)$, but $I(A, B, C), I(A, C, B), I(B, C, A)$; hence you cannot direct the links without violating the independences found by the test. Then a solution may be to remove one link and direct the remaining as a chain.

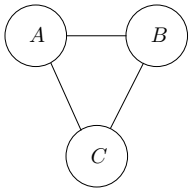


Fig. 7.9. The tests yield all pairs dependent, but all pairs independent given the third variable.

This is called an *uncertain region*: removal of a link is dependent on how you treat the other links. Note that for this example, the PC algorithm will stop after $I(A, B, C)$ and $I(A, C, B)$ and removal of the links $A-B$ and $A-C$. If the necessary path condition is used, the process will stop after $I(A, B, C)$ and removal of the link $A - B$.

There may be other reasons why it is not possible to direct links without violating some of the independences returned by the tests. Assume you have the four variables A, B, C, D, and you get the independences $I(A, C), I(A, D)$, and $I(B, D)$ for $i = 0$. Then the PC algorithm extended with the necessary path condition will stop, and you have the skeleton in Figure 7.10. Now there is no proper way of directing the links.

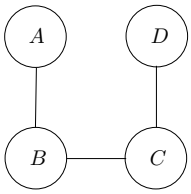


Fig. 7.10. A skeleton that cannot be directed.

Rule 1 grants the introduction of two v-structures, $A \rightarrow B \leftarrow C$ and $B \rightarrow C \leftarrow D$; but then the link $B - C$ receives two directions. For this particular case, the inconsistency need not be due to the test, but it can be caused by a hidden variable as illustrated in Figure 7.11.

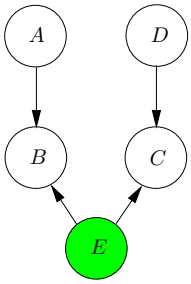


Fig. 7.11. The problem illustrated in Figure 7.10 may be caused by a hidden variable (E).

You cannot always assume that a problem related to directing the skeleton is due to erroneous tests or hidden variables. It may happen that the cases have not been sampled from a Bayesian network. Anyhow, you have to enforce directions inconsistent with the test results. Beware that violating dependence results makes it impossible to represent the joint probability distribution of the case set.

It is tempting to conclude that the PC algorithm discovers causality from observed data. for a century it has been a commonly accepted view that causality can be discovered only through controlled experiments, where an outside agent fixes some variables to certain states. The new algorithms for learning Bayesian network structures have questioned this view. The PC algorithm (and other preceding constraint-based algorithms) works on observed nonmanipulated data, and it allows you to introduce v-structures. However, you can conclude that you have discovered a causal relation only if you can be sure that there are no hidden variables obscuring the picture.

For example, consider the structure in Figure 7.12 with D and E hidden. The PC algorithm will yield $I(A, C)$, $\neg I(A, B)$, $\neg I(B, C)$ and stop. However, A and C are not causes of B. We shall not go deeper into this very lively and interesting discussion.

Finally, it shall be mentioned that even a completely correct statistical test for independence may not provide the correct d-separation properties (even if you have a very large database); the conditional probabilities in the network may hide dependencies. Take for example two switches A and B for the light C. The light is on if and only if A and B are in the same position. The prior probabilities for A and B are even. Although both links in this example

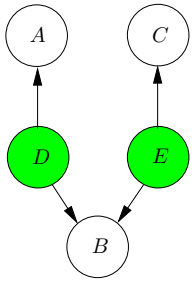


Fig. 7.12. A structure with *confounding variables*:: D and E are hidden and obscure the learning of causality.

are essential, then for a fair sample $\mathcal{D}$ we have $I_{\mathcal{D}}(A, C)$ and $I_{\mathcal{D}}(B, C)$; the problem is that the faithfulness assumption is violated.

7.2 Ockham's Razor

When learning structure from experiments, there is a general principle of inductive learning, called *Ockham's razor* (after William of Ockham, 1285–1349). It recommends that one choose the simplest hypothesis consistent with the observations.

In the case of learning Bayesian networks, this principle has a justification of its own. The complexity of a Bayesian network can be measured in number of links or in number of independent parameters.

Proposition 7.1. *Let M be a Bayesian network over the variables $\mathcal{U}$, and assume that the parameters $\boldsymbol{\theta}_M$ are both locally and globally independent (see Section 6.3.1). Then the number of independent parameters (or the* size *of M) is given by:*

$$\text{size}(M) = \sum_{X \in \mathcal{U}} |\text{pa}(X)| \cdot (|\text{sp}(X)| - 1). \tag{7.3}$$

For example, assuming that all variables are binary, then the size of the model in Figure 7.12 is $1+2+1+4+2 = 10$. On the other hand, when the assumption about either local and global parameter independence is violated, then the number of independent parameters is usually lower.

Proposition 7.2. *Let N be a Bayesian network over $\mathcal{U}$ with only essential links. Then no other Bayesian network M representing $P_N(\mathcal{U})$ can have fewer links or a smaller size than N.*

Proof. Let M represent $P(\mathcal{U})$. Since all links are essential, it must hold that whenever A and B are linked in N they are also in M. If there is a chance

for M to have smaller size than N, then it must be because some links in M have the direction opposite to that of the corresponding links in N.

Let L be a link from A to C, which is reversed. For simplicity we assume that C has only one parent more, and that A has a single parent (See Figure 7.13(a)). Figure 7.13(b) depicts the situation in which the link has been reversed.

In Figure 7.13(b) we have that C and D are independent, and A and B are independent given C. To compensate for this, M must have extra links. The cheapest will be to add a link from C to D and from B to A (See Figure 7.13(c)). Elementary arithmetic (see Exercise 7.6) now yields that the size of the Bayesian network in Figure 7.13(c) is larger than for the Bayesian network in Figure 7.13(a). Equality is only possible if A has no parents, and A is the only parent of C. □

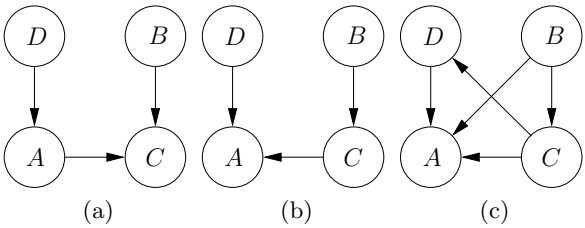


Fig. 7.13. In (a), C and D are dependent. If the link between A and C is reversed, then C and D become independent (b), and to compensate for this you can insert extra links (c).

Note that the proposition does not hold if we count probability parameters rather than size. Note also that we use conditional independence properties rather than probabilities in the proof.

The proposition justifies a search for minimal models: If the real world is a Bayesian network (with all links essential), and if the sample set is faithful, then among all the models representing the distribution, the true one is minimal with respect to links as well as size.

7.3 Score-Based Learning

When doing structural learning, we look for a Bayesian network structure that on the one hand can represent our database sufficiently well (when augmented with a set of probabilities) and on the other hand is not overly complex. In Section 7.1 we saw how to perform structural learning based on independence tests, and in this section we shall focus on another type of learning, called *score-based learning*. Score-based learning assigns a number (a score) to each

Bayesian network structure. The score reflects the "usefulness" of a structure, where the term "usefulness" can for example cover how likely it is that the structure could have been used to generate the database at hand.

If we have a score function that takes a Bayesian network structure as argument and returns a value, then the task of score-based learning can be considered a search problem: we simply look for the model structure with the highest score. This also means that a score based learning algorithm can in principle be completely described by specifying two components, (1) a score function, and (2) a search procedure.

7.3.1 Score Functions

When specifying a score for a network structure S with respect to a database $\mathcal{D}$, your first attempt might be to consider the Euclidean distance (see Definition 6.1) between the probability distribution, $P_D^{\#}(\mathcal{U})$, represented by the database $\mathcal{D}$ and your "best shot" at the probability distribution that can be encoded in S over the same set of variables. By "best shot" we mean the conditional probabilities for S that bring $P_S(\mathcal{U})$ closest to $P_{\mathcal{D}}^{\#}(\mathcal{U})$. An immediate attempt might be to use the maximum likelihood estimates $\hat{\boldsymbol{\theta}}_S$ (see Section 6.1), in which case the distance measure can be specified as

$$\text{dist}\left(P_D(\mathcal{U}), P_S(\mathcal{U} \mid \hat{\boldsymbol{\theta}})\right) = \sum_{\mathbf{x} \in \text{sp}(\mathcal{U})} \left(P_D(\mathbf{x}) - P_S(\mathbf{x} \mid \hat{\boldsymbol{\theta}}_S)\right)^2.$$

Unfortunately, there are (at least) two rather severe problems in using this distance as the score of a structure. First of all, since a complete network structure can encode any probability distribution, we know that in order to minimize the distance above, we should simply select any complete network structure; this is obviously not satisfactory. To avoid this problem you could augment the score with a term penalizing model complexity (see Proposition 7.2). This means that the score of a structure should be defined as a trade-off between how good it is at representing the distribution encoded by the database and the complexity/size of the structure. A possible suggestion for such a score could then be

$$\text{score}\left(P_D(\mathcal{U}), P_S(\mathcal{U} \mid \hat{\boldsymbol{\theta}})\right) = \text{dist}\left(P_D(\mathcal{U}), P_S(\mathcal{U} \mid \hat{\boldsymbol{\theta}})\right) + c \cdot \text{size}(S),$$

where c is a (user specified) constant used to control the trade-off between model accuracy and model complexity.

However, even though we may have found a suggestion for a score function that reliably reflects the usefulness of a structure, we still have another problem to address: from a computational perspective, the Euclidean distance can be extremely difficult to work with, since it is a function of $P_D^{\#}(\mathcal{U})$. That is, it basically requires us to deal with a table over the joint state space of all the variables, and we are therefore faced with the same combinatorial explosion, which we again and again try to avoid.

To summarize the discussion above, we look for a score function that should (at least) have the following properties:

- It should balance the accuracy of a structure with the complexity of the structure.
- It should be computationally tractable to evaluate.

The Bayesian Information Criterion

An example of a score function satisfying the above two properties is the *Bayesian information criterion* (BIC), which contains a term measuring how well the data fits the model as well as a term that accounts for model complexity:[1]

$$\mathrm{BIC}(S \mid \mathcal{D}) = \log_2 P(\mathcal{D} \mid \hat{\theta}_S, S) - \frac{\mathrm{size}(S)}{2} \log_2(N), \tag{7.4}$$

where $\hat{\boldsymbol{\theta}}$ is an estimate of the maximum likelihood parameters for the structure S. If we furthermore assume that the cases are independent given the model, then

$$\mathrm{BIC}(S \mid \mathcal{D}) = \sum_{i=1}^{N} \log_2 P(\mathbf{d}_i \mid \hat{\theta}_S, S) - \frac{\mathrm{size}(S)}{2} \log_2(N).$$

In order to score a model using BIC, you start off by estimating the maximum likelihood parameters for the model. If the database is complete, then this is just a matter of frequency counting, but if some of the cases contain missing values you may run the EM algorithm. Based on these estimates you calculate the probability for each case in the database. This can be done by simply inserting the case as evidence in the Bayesian network and performing a propagation; the probability of the case is then the probability of the evidence. If all the cases are complete, then this task is even simpler, you just multiply the appropriate entries in the conditional probability tables, which, in turn, are frequency counts derived from the database! This also means that the calculation of the BIC score has been reduced to a counting problem: let r_i denote the number of states for variable X_i, and let $q_i = \prod_{X_l \in \Pi_i} r_l$ denote the number of configurations over the *parents* for X_i in S (if X_i does not have any parents then we let $q_i = 1$). With this notation we now have (the derivation is left as an exercise)

$$\mathrm{BIC}(S \mid \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log_2 \left(\frac{N_{ijk}}{N_{ij}} \right) - \frac{\log_2 N}{2} \sum_{i=1}^{n} q_i (r_i - 1), \tag{7.5}$$

where N_{ijk} denotes the number of cases in the database with X_i in its kth configuration and $\mathrm{pa}(X_i)$ in the jth configuration.

[1] The exact form of the BIC score can be derived from a Taylor expansion of $P(\mathcal{D} \mid S)$.

Example 7.2. Consider the two Bayesian network structures over the two binary variables X_1 and X_2 shown in Figure 7.14 (we shall refer to them as B_a and B_b, respectively), and assume that we have the database shown in Table 7.2.

(a) Complete dependence; (b) complete independence.

Fig. 7.14. Two BN model structures for the domain $\mathbf{X} = (X_1, X_2)$.

Case	X_1	X_2
1.	*yes*	*positive*
2.	*yes*	*positive*
3.	*yes*	*positive*
4.	*yes*	*positive*
5.	*yes*	*positive*
6.	*yes*	*positive*
7.	*yes*	*negative*
8.	*yes*	*negative*
9.	*no*	*negative*
10.	*no*	*negative*

	yes	*no*
X_1	8	2

		X_1	
		yes	*no*
X_2	*pos*	6	0
	neg	2	2

Table 7.2. A database for the two binary variables X_1 and X_2 as well as the counts N_{11k} and N_{2jk} derived from the database.

In order to calculate the BIC score for B_a we first calculate the counts (the states *yes* and *positive* correspond to state number 1) shown in Table 7.2. By substituting these values into equation (7.5), we get

$$\begin{aligned}
&\mathrm{BIC}(S \,|\, \mathcal{D}) \\
&\quad = \left[8 \cdot \log\left(\frac{8}{8+2}\right) + 2 \cdot \log\left(\frac{2}{8+2}\right) + 6 \cdot \log\left(\frac{6}{6+2}\right) + 2 \cdot \log\left(\frac{2}{6+2}\right)\right. \\
&\qquad \left. + 0 \cdot \log\left(\frac{0}{0+2}\right) + 2 \cdot \log\left(\frac{2}{0+2}\right)\right] - \frac{1+2}{2} \log(10) \\
&\quad = -18.69.
\end{aligned}$$

For the network BN_b we calculate the following counts, which can be read and derived from the counts in Table 7.2: $N'_{111} = 8$, $N'_{112} = 2$, $N'_{211} = N_{211} + N_{221} = 6$ and $N'_{212} = N_{212} + N_{222} = 4$. This gives us

$$
\begin{aligned}
&\mathrm{BIC}(S \mid \mathcal{D}) \\
&\quad = \left[8 \cdot \log\left(\frac{8}{8+2}\right) + 2 \cdot \log\left(\frac{2}{8+2}\right) + 6 \cdot \log\left(\frac{6}{6+4}\right) + 6 \cdot \log\left(\frac{4}{6+4}\right)\right] \\
&\qquad - \frac{1+1}{2}\log(10) \\
&\quad = -20.25.
\end{aligned}
$$

That is, according to the BIC score we should choose B_a rather than B_b.

7.3.2 Search Procedures

Given a score function, the task is to find the highest-scoring Bayesian network structure among the set of all possible network structures. That is, the task of structural learning has been reduced to a search problem. The challenging part of this problem is that the size of the space of all structures is super-exponential in the number of nodes (see equation (7.1)) so an exhaustive enumeration of all the structures is not possible.

Instead, researchers have considered heuristic search strategies that move around in the search space by iteratively performing small changes to the current structure. Most commonly, these search methods work directly on the space of Bayesian network structures; hence each point in such a *search space* corresponds to a particular DAG; in the remainder of this section we shall use the terms structure and DAG interchangeably, since the state spaces of the variables in the structure is fixed.

The definition of the search space determines the definition of the *search operators* used to move from one structure to another. In turn, these operators determine the *neighborhood* of a DAG, namely the DAGs that can be reached in one step from the current DAG. Typically, the operators consist of:

- *arc addition*: insert a single arc between two nonadjacent nodes.
- *arc deletion*: remove a single arc between two nodes.
- *arc reversal*: reverse the direction of a single arc.

In what follows we let $op(S, A)$ represent the result of performing the arc operation A on the structure S, i.e., $op(S, A)$ is a DAG that differs from S with respect to one arc.

One important property of these operators is that they result only in local changes to the current structure; for example, if an arc is inserted from node X_i to X_j, then only the family of node X_j is changed, and similarly if an arc is deleted; if an arc is reversed, then the families of both X_i and X_j are changed. This property can be exploited when we have a so-called decomposable score function.

Definition 7.3. *A score function is said to be* decomposable *if it can be expressed as a sum of local scores, one for each family of nodes in the structure:*

$$\text{score}(\mathcal{D}, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), \mathcal{D}).$$

The BIC score is an example of a decomposable score function for complete data, since it can be written as

$$\text{BIC}(S \mid \mathcal{D}) = \sum_{i=1}^{n} \left[\sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \left(\frac{N_{ijk}}{N_{ij}} \right) - \frac{1}{2} q_i (r_i - 1) \log N \right].$$

This decomposition property can be used when we evaluate the benefit of making an arc change. For example, if we insert an arc from X_i to X_j, then only the local score for X_j will change, i.e., when evaluating whether such a move is beneficial we need to evaluate only the score difference (or gain)

$$\Delta(X_i \rightarrow X_j) = \text{score}(X_j, \text{pa}(X_j) \cup \{X_i\}, \mathcal{D}) - \text{score}(X_j, \text{pa}(X_j), \mathcal{D}). \quad (7.6)$$

Greedy Search

A simple heuristic search procedure is greedy search: choose some initial structure (usually the empty structure, a randomly chosen structure, or a prior structure specified by the user) and calculate the gain for each legal arc operation; by legal we mean that the resulting graph should be acyclic. Next, perform the arc operation A with highest gain (if positive) and use the resulting model as your current model. More formally:

Algorithm 7.2 [Greedy search]

1. *Let S be an initial structure.*
2. *Repeat*
   a) *Calculate $\Delta(A)$ for each legal arc operation A*
      - *Let $\Delta^* = \max_A \Delta(A)$ and $A^* = \arg\max_A \Delta(A)$.*
   b) *If $\Delta^* > 0$, then*
      - *Set $S = op(S, A^*)$.*
3. *Until $\Delta^* \leq 0$.*

□

It should be noted that in the greedy algorithm above you can further exploit the decomposition property of the score function: If the parents sets of two nodes, say X_i and X_j, do not change from one iteration to another, then the gain (equation (7.6)) of any arc operation involving X_i and X_j will remain unchanged. This gain can therefore be cached for subsequent iterations so that the calculations can be reused.

Obviously, when we work with heuristic search algorithms we are not guaranteed to find a global optimal structure but only a local optimal structure. Several methods have been proposed to escape local maxima. An example of

this is greedy search with multiple restarts: after a local maximum is found the search is reinitialized with a random structure. This reinitialization is then repeated for a fixed number of iterations, and the best structure found throughout the entire process is selected.

Prior Information

A way of reducing the search space (and thereby also the risk of ending up in a local maximum) is to incorporate prior information, thus constraining the models under investigation.

There are various standard ways of constraining the models to consider. First, causality can be exploited. If possible, the nodes are clustered in a causal hierarchy. You may, for example, consider a medical domain in which you have disease nodes **D**, symptom nodes **S**, risk factor nodes **R**, and treatment nodes **T**. Then, you need not consider links from a node in **S** to a node in **T**. The full hierarchy is shown in Figure 7.15.

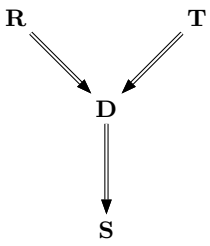


Fig. 7.15. A causal hierarchy for clusters of nodes. Directed links are allowed only inside a cluster or downward in the hierarchy.

If **R** and **T** have two nodes, and **D** and **S** have three nodes, then the hierarchy allows approximately 10^{15} different DAGs. This is a considerable reduction compared to $4.2 \cdot 10^{18}$, but still it is extremely many. This prior knowledge could then be included directly in the search algorithm by considering an arc operation as being legal only if it adheres to the causal hierarchy.

A more general approach could be to specify a partial ordering, $\preceq$, over the variables, such that we allow an arc from X_i to X_j only if $X_i \preceq X_j$. In the special case that we have a linear ordering, then the i'th node can have at most $i - 1$ parents producing 2^{i-1} different parent sets. The number of structures consistent with the ordering is therefore

$$\prod_{i=1}^{n} 2^{n-1} = 2^{\sum_{i=1}^{n-1} i} = 2^{n(n-1)/2}.$$

Although the number of structures is still exponential, specifying a linear ordering provides a substantial reduction. For example, with 10 nodes we have

$3.5 \cdot 10^{13}$ different structures as opposed to $4.2 \cdot 10^{18}$ in the unrestricted case. Whether it is reasonable to specify such an ordering is heavily dependent on the domain in question. However, you could imagine different rules of thumb. For instance, if the variables represent events that manifest themselves at different points in time, then you may be able to order the variables according to these time points. An example of this could be variables representing components in a physical production process, where there is a time delay for an item to move from one component to another.

Finally, you could also use more specific expert statements when reducing the model space. All positive as well as negative statements on the presence of links reduce the number by a factor between 2 and 3. Consider again the medical domain above. If, for example, the expert states that the nodes in **D** are independent given **R** and **T** (three links missing), then the model space is reduced by a factor of 25.

*Equivalence Class Search

It can sometimes be advantageous to define the search space using a more abstract representation than DAGs. An example is a procedure called *greedy equivalence search*.

The search is based on the observation that data alone cannot be used to discriminate among structures with the same d-separation properties (see also Section 7.1.1).

Definition 7.4. *Two network structures B_1 and B_2 are said to be* equivalent *if they have the same d-separation properties.*

The equivalence relation is reflexive, symmetric, and transitive; hence the relation defines a collection of *equivalence classes*.

A score function assigning the same score to equivalent structures is said to be *score equivalent*; the BIC score is an example of a score-equivalent function (see Figure 7.16). This also means that if we have identified a particular structure using a score equivalent function, then we could just as well pick any other structure equivalent to the one identified. A way of making this observation explicit is to define the search space such that each point in the search space corresponds to an equivalence class.

In order to move around in the space of equivalence classes, we should also specify a set of search operators, but due to the nature of the search space, these operators are a bit more complex than the ones used in DAG spaces. Instead we shall only define the *neighborhood* for an equivalence class: the set of structures reachable by a single change to the current structure or one of its equivalents. Since the equivalence classes are defined in terms of independence statements, we define the neighborhood of an equivalence class in this way. We have an *upper neighborhood* consisting of equivalence classes with fewer dependence statements, and a *lower neighborhood* with more dependence statements. The two neighborhoods are defined as the equivalence

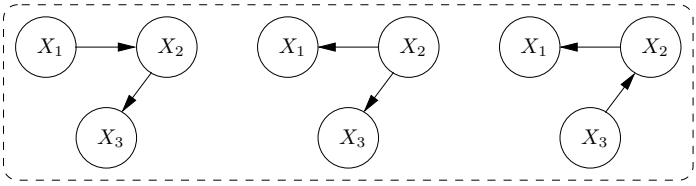


Fig. 7.16. The three DAGs constitute an equivalence class, and any score equivalent scoring function will assign the same score to all three structures.

classes that can be obtained by either adding or deleting a single arc from a DAG in the current equivalence class. Figure 7.17 illustrates the different equivalence classes for three variables; the arcs attached to an equivalence class identify the upper and lower neighborhoods.

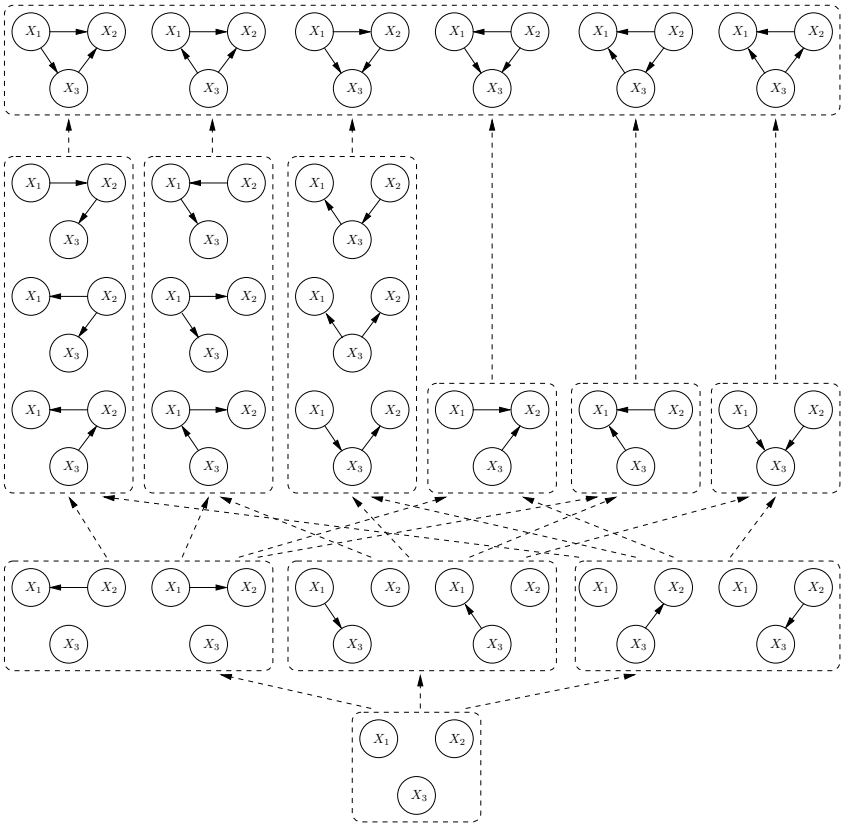


Fig. 7.17. The equivalence class hierarchy for the possible structures over three variables.

Based on this specification of the search space, the greedy equivalence search algorithm consists of two steps: First, start with the equivalence class representing no dependencies among the variables (the bottom equivalence class in Figure 7.17), and perform a greedy search upward until a local maximum is reached. Next, starting from the equivalence class just identified, perform a greedy search downward until a local maximum is reached. It has been proved that if the database is sufficiently large, then the resulting equivalence class is guaranteed to include the Bayesian network from which the data was generated.

Finally, it should be emphasized that even though we have made another specification of the search space, we have unfortunately not solved the general complexity problem that we faced in DAG spaces: the number of equivalence classes also grows super-exponentially in the number of variables.

7.3.3 Chow–Liu Trees

The BIC score function incorporates a penalty term to control model complexity. Another way of dealing with this issue is to put restrictions on the allowable network structures so that overly complex structures are not considered. A particular simple class of Bayesian network structures is the set of tree-shaped structures, where each node is allowed at most one parent (see Figure 7.18). Not only is probability updating very easy in these networks, but Chow and Liu also showed that a network of maximal likelihood can be learned efficiently from a database; due to this result, these tree structures are also called *Chow–Liu trees*.

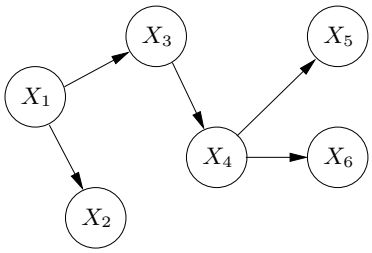


Fig. 7.18. An example of a Chow–Liu tree structure.

Theorem 7.2 (Learning of Chow-Liu trees). *Let $\mathcal{D}$ be a data set over the variables $\{X_1, \ldots, X_n\}$. A Chow-Liu tree of maximal likelihood can be constructed as follows:*

1. *Calculate the mutual information* $\mathrm{MI}(X_i, X_j)$ *for each pair* (X_i, X_j).
2. *Consider the complete MI-weighted graph: the complete undirected graph over* $\{X, \ldots, X_n\}$, *where the links* (X_i, X_j) *have the weight* $\mathrm{MI}(X_i, X_j)$.

3. *Build a maximal-weight spanning tree for the complete MI-weighted graph.*
4. *Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.*
5. *Learn the parameters.*

Notes:

- The likelihood of a Bayesian network B given a data set $\mathcal{D}$ is the same as described in Section 7.3.1: $P(\mathcal{D}|B)$.
- The formula for mutual information is

$$\mathrm{MI}(X,Y) = \sum_{X,Y} P(X,Y) \log_2 \left(\frac{P(X,Y)}{P(X)P(Y)} \right).$$

- A maximal-weight spanning tree can be constructed through Kruskal's algorithm: choose repeatedly a link of maximal weight not producing a cycle.
- Calculating the mutual information for a pair of variables requires one sweep through the data. If the database consists of N cases, then this can be done in time $O(N)$, and since we need to perform this calculation for all pairs of variables, the overall time complexity of the Chow–Liu algorithm becomes $O(n^2 \cdot N)$.

Example 7.3. Consider the *Cold or Angina* problem described in Section 3.1.2 and assume that we have a database of cases from this domain. For simplicity we assume that the cases are a faithful sample from the model in Figure 3.6, with the probabilities specified as in Section 3.2.5, Table 3.15 (Page 76) and Table 3.20 (Page 96).

In order to learn a Chow–Liu tree for this domain, we start by calculating the mutual information between each pair of variables (the following calculations are based on the specified model). For the variables *Cold* and *SoreThroat?* (*Sore*) we get

$$\begin{aligned} \mathrm{MI}(\mathit{Cold}, \mathit{Sore}) &= \sum_{\mathit{Cold},\mathit{Sore}} P(\mathit{Cold}, \mathit{Sore}) \log_2 \left(\frac{P(\mathit{Cold}, \mathit{Sore})}{P(\mathit{Cold})P(\mathit{Sore})} \right) \\ &= 0.02101216. \end{aligned}$$

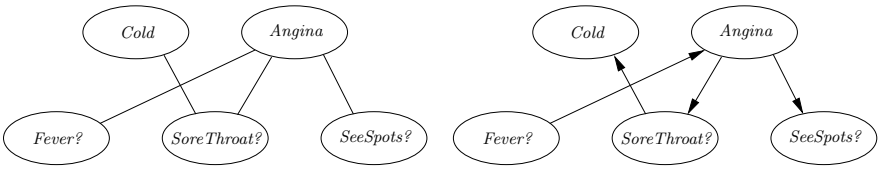
The mutual information for all pairs of variables is given in Table 7.3.

Based on these calculations we can construct a maximal-weight spanning tree by starting from the empty graph, and iteratively adding an edge with maximum weight as long as no cycle is created. The resulting structure is shown in Figure 7.19(a), and by picking *Fever?* as a root and directing the edges away from *Fever?* we obtain the Chow–Liu tree in Figure 7.19(b).

Since the learned model has a tree structure, the model may specify (conditional) independences that are not reflected in the data. On the other hand,

MI(*Cold*, *Angina*) = 0	MI(*Fever?*, *Angina*) = 0.015076
MI(*SoreThroat?*, *Angina*) = 0.018016	MI(*SeeSpots?*, *Angina*) = 0.0180588
MI(*Cold*, *Fever?*) = 0.014392	MI(*Cold*, *SoreThroat?*) = 0.0210122
MI(*Cold*, *SeeSpots?*) = 0	MI(*SoreThroat*, *Fever?*) = 0.0015214
MI(*Fever?*, *SeeSpots?*) = 0.0017066	MI(*SeeSpots?*, *SoreThroat?*) = 0.0070697

Table 7.3. The mutual information for each pairs of variables.



(a) (b)

Fig. 7.19. Figure (a) shows a maximal weight spanning tree based on the MI-calculations in Table 7.3. Figure (b) shows the Chow-Liu tree obtained by selecting the variable *Fever?* as root and directing the edges away from *Fever?*.

even though the independence properties are inaccurate, it has turned out that the model may still provide a good approximation. We shall return to this issue in Chapter 8, where we will use Chow–Liu trees in a classification context.

Proof. [Learning of Chow–Liu trees, Theorem 7.2]

The proof involves some pencil pushing. First we rewrite the log-likelihood of a Bayesian network B given the data $\mathcal{D} = (\mathbf{d}_1, \ldots, \mathbf{d}_N)$:

$$\log_2 P(\mathcal{D} \mid B) = \log_2 \prod_{j=1}^{N} P(\mathbf{d}_j \mid B) = \sum_{j=1}^{N} \log_2 P(\mathbf{d}_j \mid B)$$
$$= \sum_{j=1}^{N} \sum_{i=1}^{n} \log_2 P(X_i = \mathbf{d}_j \mid \mathrm{pa}(X_i) = \mathbf{d}_j, B).$$

The number of cases in $\mathcal{D}$ that agree on a particular configuration of X_i and $\mathrm{pa}(X_i)$ is given by $N(X_i, \mathrm{pa}(X_i)) = N \cdot P^{\#}(X_i, \mathrm{pa}(X_i))$. Hence instead of summing over all the cases, we can write

$$\log_2 P(\mathcal{D} \mid B) = N \cdot \sum_{i=1}^{n} \sum_{X_i, \mathrm{pa}(X_i)} P^{\#}(X_i, \mathrm{pa}(X_i)) \cdot \log_2 P(X_i \mid \mathrm{pa}(X_i), B).$$

Since we are looking for a Bayesian network of maximal likelihood, we can assume that the parameters of B are maximum likelihood parameters (see Section 6.1.1), i.e., $P(X \mid \mathrm{pa}(X), B) = P^{\#}(X \mid \mathrm{pa}(X))$, and therefore

$$\log_2 P(\mathcal{D} \mid B) = N \cdot \sum_{i=1}^{n} \sum_{X_i, \mathrm{pa}(X_i)} P^{\#}(X_i, \mathrm{pa}(X_i)) \cdot \log_2 P^{\#}(X_i \mid \mathrm{pa}(X_i)).$$

This equation can be rewritten as

$$\log_2 P(\mathcal{D} \mid B) = N \cdot \sum_{i=1}^{n} \sum_{X_i, \mathrm{pa}(X_i)} P^{\#}(X_i, \mathrm{pa}(X_i)) \cdot \left(\log_2 \frac{P^{\#}(X_i, \mathrm{pa}(X_i))}{P^{\#}(X_i) P^{\#}(\mathrm{pa}(X_i))} + \log_2 P^{\#}(X_i) \right),$$

and since the parent sets contain at most one variable, we get

$$\log_2 P(\mathcal{D} \mid B) = N \cdot \sum_{i=1}^{n} \mathrm{MI}(X_i, \mathrm{pa}(X_i)) + \sum_{i=1}^{n} \sum_{X_i} P^{\#}(X_i) \cdot \log_2 P^{\#}(X_i).$$

This expression is maximized by choosing parents such that the sum of the MI terms is maximized. Since B should be a tree and each parent set contains at most one variable, step three in the theorem is guaranteed to maximize the log-likelihood. Finally, by choosing an arbitrary root and directing the arcs away from the root, we ensure that each node will get at most one parent, and from the d-separation properties we also see that we get the same independence properties regardless of the choice of root. □

7.3.4 *Bayesian Score Functions

The BIC score is an example of a score function combining a maximum likelihood term with a term measuring complexity. Another approach for measuring the fitness of a Bayesian network model structure, S, is to calculate the posterior probability that the data was generated by a distribution with the same independence properties as S. If we abuse the notation slightly, and also use S to denote the hypothesis that the data is sampled from a distribution with the same independence properties as S, then we have:

$$P(S \mid \mathcal{D}) = \frac{P(\mathcal{D}, S)}{P(\mathcal{D})} = \frac{P(S) P(\mathcal{D} \mid S)}{P(\mathcal{D})} = \mu P(S) P(\mathcal{D} \mid S), \tag{7.7}$$

where $\mu = P(\mathcal{D})$ is the normalization constant. This constant does not depend on S, and it is therefore not necessary to calculate it when we compare two network structures. Actually, if you were to calculate $P(\mathcal{D})$ you would be faced with a computational problem, because the calculation of this constant involves summing over all possible model structures, i.e., $P(\mathcal{D}) = \sum_B P(B) P(\mathcal{D} \mid B)$.

From equation (7.7) we see that in order to score a structure based on its posterior probability given the data, we only need two terms, namely the prior

probability of the structure ($P(S)$) and the *marginal likelihood* of the structure given the data ($P(\mathcal{D} \mid S)$). Typically you would choose a prior probability distribution for the structures that is relatively easy to calculate, and the main computational problem is therefore the calculation of the marginal likelihood, where we will have to deal with the parameters of the model $\boldsymbol{\theta}_S$ (we shall return to the specification of structure priors in Section 7.3.4):

$$P(\mathcal{D}|S) = \int_{\boldsymbol{\theta}_S} P(\mathcal{D}|S, \boldsymbol{\theta}_S) f(\boldsymbol{\theta}_S|S) d\boldsymbol{\theta}_S, \tag{7.8}$$

where $f(\boldsymbol{\theta}_S \mid S)$ is a prior probability distribution over the parameters (conditional probabilities) for S. The integral in the above equation is over all parameters, and, in effect, over all possible Bayesian networks with the same structure but with different conditional probability distributions. Intuitively, the marginal likelihood can therefore be interpreted as the probability that we could generate the database $\mathcal{D}$ if we were to randomly select the parameters for S according to the parameter prior $f(\boldsymbol{\theta}_S \mid S)$.

As hinted above, the hard part in the calculation of $P(S \mid \mathcal{D})$ is the evaluation of the integral in equation (7.8). Fortunately, it has been shown that the evaluation of this integral can be reduced to a simple counting problem based on the following six assumptions:

1. the database $\mathcal{D}$ is a faithful sample from some Bayesian network;
2. the cases in the database $\mathcal{D}$ are independent given the BN model;
3. the database is complete;
4. the prior distribution of the parameters in every Bayesian network is uniform;
5. [local independence] for any two configurations over the parents for a variable X_i, the parameters for the conditional probability distributions associated with X_i are independent; and
6. [global independence] the densities of the parameters for the conditional probability distributions for X_i and X_j are independent for $i \neq j$.

Now let us again use N_{ijk} to denote the number of cases in the database that include the configuration $(X_i = k, \mathrm{pa}(X_i) = j)$. Based on the assumptions above, the following theorem has been proved.

Theorem 7.3. *Let $\mathcal{D}$ be a database over the variables $X_1, X_2, \ldots, X_n$, and consider the Bayesian network structure B_s over the same set of variables. Given the six assumptions above, it holds that*

$$P(\mathcal{D} \mid S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_1 - 1)!} \prod_{k=1}^{r_i} (N_{ijk})! \quad , \tag{7.9}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

This means that the evaluation of the integral in equation (7.8) is reduced to a counting problem, which can be carried out in polynomial time.

Example 7.4. Consider again the two Bayesian network structures from Fig. 7.14, and assume that we have the database from Table 7.2 for the two binary variables X_1 and X_2.

Let us also assume that we have a priori the same belief in the two network structures, $P(BN_a) = P(BN_b)$. In order to select between B_a and B_b, the task then reduces to the calculation of

$$P(\mathcal{D}|S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_1 - 1)!} \prod_{k=1}^{r_i} (N_{ijk})!$$

for both networks. To start off, consider network B_a ($X_1 \rightarrow X_2$). As for the calculation of the BIC score (see Example 7.2), we also need the following counts, which can be found from the database: $N_{111} = 8$, $N_{112} = 2$, $N_{211} = 6$, $N_{212} = 2$, $N_{221} = 0$, and $N_{222} = 2$. By using these values we get

$$P(\mathcal{D}|BN_a) = \frac{(2-1)!8!2!(2-1)!6!2!(2-1)!2!0!}{(10+2-1)!(8+2-1)!(2+2-1)!} = 2.67 \cdot 10^{-6}.$$

For the network BN_b we have $N_{111} = 8$, $N_{112} = 2$, $N_{211} = 6$, and $N_{212} = 4$. This gives us

$$P(\mathcal{D}|BN_b) = \frac{(2-1)!8!2!(2-1)!6!4!}{(10+2-1)!(10+2-1)!} = 8.75 \cdot 10^{-7}.$$

So with a uniform prior distribution over both structure and parameters we should prefer B_a over B_b.

Although the metric provides a simple expression for calculating the likelihood of a structure, it also rests on assumptions that may not always be appropriate. Most notably, it requires that the database be complete. When these assumptions are not fulfilled you will have to resort to other methods, such as the BIC score or constraint-based algorithms.

Prior Distribution over Structures

In order to score a network structure using the metric above, you have to specify a prior distribution $P(S)$ over the network structures. The specification of this prior can be used to guide the subsequent structure search, although the contribution from the prior distribution is usually dominated by the likelihood term $P(\mathcal{D} \mid S)$ when the database gets large ($P(\mathcal{D} \mid S)$ decreases exponentially fast as cases are added to the database). One exception, however, occurs when some of the network structures are given zero probability a priori, in which case the data cannot change that belief.

Common to most (if not all) prior distributions over structures currently used is that they can be expressed as a product (or sum) with one term for each family of nodes in the network:

$$P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)),$$

where c is a normalization constant that does not depend on S. These types of prior probability distributions are *decomposable*, which means that equation (7.9) is also decomposable (see Definition 7.3).

The simplest prior distribution is the one that encodes complete ignorance, i.e., we use an even distribution over the possible network structures:

$$\rho(X_i, \text{pa}(X_i)) = 1.$$

A more informative prior that has been suggested is based on the difference between the families in the current network S and the families in a user-specified prior network B_p. Specifically, let δ_i denote the number of parents that B_P and S disagree on for X_i:

$$\delta_i = \left| (\text{pa}(X_i)_S \cup \text{pa}(X_i)_{B_P}) \setminus (\text{pa}(X_i)_S \cap \text{pa}(X_i)_{B_P}) \right|. \tag{7.10}$$

Then we can give a low prior probability to structures that are far away from the prior network structure, B_P, by setting

$$\rho(X_i, \text{pa}(X_i)) = \kappa^{\sum_{i=1}^{n} \delta_i},$$

where $0 < \kappa \leq 1$ is a user-specified constant.

Example 7.5. Consider the four Bayesian network structures depicted in Fig. 7.20, and assume that Figure 7.20(B_P) is a prior network specified by the user.

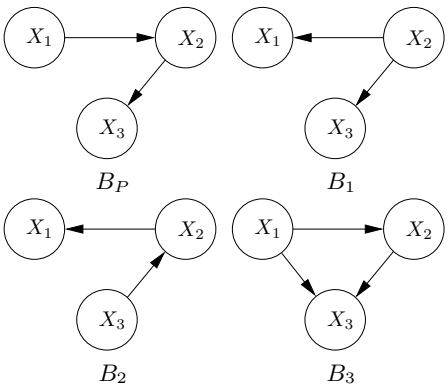


Fig. 7.20. The candidate structures B_1, B_2, and B_3 are assigned a prior distribution based on the number of arcs they have in common with the prior network B_P.

When using equation (7.10) to calculate the unnormalized prior probability for the candidate structure B_1, we first calculate the differences in the node families:

$$\delta_1^{B_1} = |(\{X_2\} \cup \emptyset) \setminus (\{X_2\} \cap \emptyset)| = 1;$$
$$\delta_2^{B_1} = |(\emptyset \cup \{X_1\}) \setminus (\emptyset \cap \{X_1\})| = 1;$$
$$\delta_3^{B_1} = |(\{X_2\} \cup \{X_2\}) \setminus (\{X_2\} \cap \{X_2\})| = 0.$$

Hence, the total difference between the two structures is measured as $\delta^{B_1} = \delta_1^{B_1} + \delta_1^{B_2} + \delta_1^{B_3} = 2$, which gives the prior probability

$$P(B_1) = c \cdot \kappa^{\delta} = c \cdot \kappa^2. \tag{7.11}$$

For B_2 we get $\delta_1^{B_2} = 1$, $\delta_2^{B_2} = 2$, and $\delta_3^{B_2} = 1$, and therefore $\delta = 4$ and $P(B_2) = c \cdot \kappa^4$. Finally, for B_3 we have $\delta_1^{B_3} = 0$, $\delta_2^{B_3} = 1$ and $\delta_3^{B_3} = 1$, and therefore $P(B_3) = c \cdot \kappa^2$. That is, a priori, B_p would be given the highest probability, then B_1 and B_3, and finally, B_2 would be given the lowest probability; observe that the normalization constant is of no importance when comparing structures.

Finally, it should be emphasized that although we can easily come up with elaborate prior distributions, there is also a caveat: the prior distribution does not necessarily assign the same score to equivalent network structures (as in Example 7.5). When this is the case, then if it is used to define, say, the score function in equation (7.9), the resulting score function is not score equivalent. As an example, consider equation (7.10), and use any prior network structure that is different from the empty graph.

Regulating Model Complexity

An attractive property of the BD score (and likelihood based scoring functions in general) is that it has an intrinsic property that no extra term is needed for penalizing complexity.

The intuition why the BD score is less likely to pick out an overfitted network structure is closely related to the Bayesian version of Ockham's razor: A complex structure with few conditional independences can generate many possible data sets, so it is unlikely that it has generated this particular data set at hand (see Figure 7.21 for an illustration). Obviously, models that are too simple are also unlikely to have generated the data.

To provide a specific example, consider again the Bayesian network structures depicted in Figure 7.14. From the model in Figure 7.14(a) you can sample a database, and then use it to score the model structure S in Figure 7.14(b), where X_1 and X_2 are independent. Specifically, let the databases be generated according to the following probability distributions: $P(X_1) = (0.5, 0.5)$,

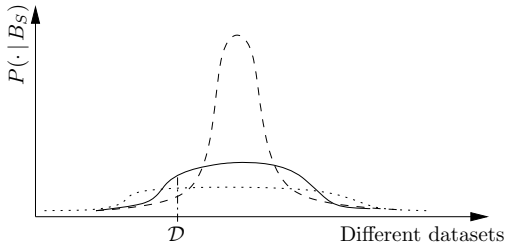


Fig. 7.21. The figure illustrates the marginal likelihood for three different structures; the dotted line represents a structure that is too complex, the dashed line represents a structure that is too simple, and the solid line represents an appropriate structure.

$$P(X_2 = 1|X_1 = 0) = 0.5(1 - \epsilon),$$
$$P(X_2 = 1|X_1 = 1) = 0.5(1 + \epsilon),$$

where the parameter ϵ varies between 0 and 1 and is used to control the strength of the dependency between X_1 and X_2; the larger the value of ϵ, the stronger the dependency. A plot of $P(S \mid \mathcal{D})$ for four different database sizes (generated for various values of ϵ) is depicted in Figure 7.22. In particular, we can see that when the database is not too large, S is acceptable even for relatively large values of ϵ. For example, with $\epsilon = 0.2$ we have that $P(S \mid \mathcal{D}) \approx 0.6$ for a database with 100 cases.

7.4 Summary

Constraint-Based Methods

The structure of a Bayesian network can be learned from independence statements of the form, "A independent of B given C" (denoted by $I(A, B, C)$):

1. Find the skeleton of the Bayesian network: the link $A - B$ is part of the skeleton if and only if $\neg I(A, B, \mathcal{X})$ for all $\mathcal{X}$ not containing A or B.
2. Direct the links:
   Introduction of v-structures: If you have three nodes, A, B, C, such that $A - C$ and $B - C$, but not $A - B$, then introduce the v-structure $A \rightarrow C \leftarrow B$ if there exists an $\mathcal{X}$ (possibly empty) such that $I(A, B, \mathcal{X})$ and $C \notin \mathcal{X}$.
   Avoid new v-structures: When Rule 1 has been exhausted, and you have $A \rightarrow C - B$ (and no link between A and B), then direct $C \rightarrow B$.
   Avoid cycles: If $A \rightarrow B$ introduces a directed cycle in the graph, then do $A \leftarrow B$.
   Choose randomly: If none of the rules 1–3 can be applied anywhere in the graph, choose an undirected link and give it an arbitrary direction.

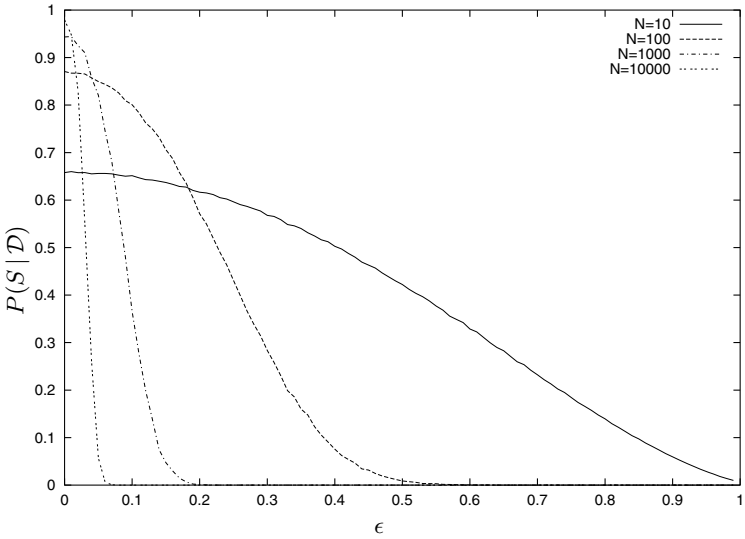


Fig. 7.22. A plot of $P(\text{complete independence} \mid \mathcal{D})$ for four different database sizes. The databases were sampled from a model with two variables, where the level of dependency between the variables was controlled by the value of ϵ (a high value implies a high dependency).

The independence statements can be established through statistical tests on a database.

The PC algorithm: The initial skeleton can be found using the PC-algorithm:

1. Start with the complete graph;
2. $i := 0$;
3. **while** a node has at least $i + 1$ neighbors
   - **for all** nodes A with at least $i + 1$ neighbors
     - **for all** neighbors B of A
       - **for all** neighbor sets $\mathcal{X}$ such that $|\mathcal{X}| = i$ and $\mathcal{X} \subseteq (\text{nb}(A) \setminus \{B\})$
         - **if** $I(A, B, \mathcal{X})$ **then** remove the link $A - B$ and store "$I(A, B, \mathcal{X})$"
   - $i := i + 1$

Score-Based Methods

A Bayesian network can be learned from a database by performing a search in the space of all DAGs and selecting the one with the highest score.

The BIC Score:

$$\mathrm{BIC}(S \mid \mathcal{D}) = \sum_{i=1}^{N} \log_2 P(\mathbf{d} \mid \hat{\theta}_S, S) - \frac{\mathrm{size}(S)}{2} \log_2(N).$$

Chow–Liu trees

A tree-shaped Bayesian network of maximal likelihood can be learned in polynomial time using the Chow–Liu algorithm:

1. Calculate the mutual information $\mathrm{MI}(X_i, X_j)$ for each pair (X_i, X_j).
2. Consider the complete MI-weighted graph: the complete undirected graph over $\{X, \ldots, X_n\}$, where the links (X_i, X_j) have the weight $\mathrm{MI}(X_i, X_j)$.
3. Build a maximal-weight spanning tree for the complete MI-weighted graph.
4. Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.
5. Learn the parameters.

Mutual information:

$$\mathrm{MI}(X, Y) = \sum_{X,Y} P(X, Y) \log_2 \left(\frac{P(X, Y)}{P(X)P(Y)} \right).$$

7.5 Bibliographical Notes

The first method for automated learning of Bayesian networks was the method of Chow and Liu (1968), which learned tree-structured models. A statistical approach for learning Bayesian networks through manually selected independence tests was given by Edwards and Havranek (1985). The PC algorithm was developed by Spirtes *et al.* (1993); see also (Spirtes *et al.*, 2000). It is an extension of work by Wermuth and Lauritzen (1990) and Verma and Pearl (1991). The necessary path condition and uncertain areas are due to Steck (2001). Improved algorithms, which theoretically should be more robust in face of flawed independence tests, are given by Margaritis and Thrun (1999) and Cheng *et al.* (2002). A discussion on observing causality can be found in (Pearl, 2000).

The dimensionality of models with hidden variables has been explored by Geiger *et al.* (1996) in the context of model selection. Here the BIC score (Schwarz, 1978) was extended to Bayesian networks with hidden variables. The BIC score of a model is an asymptotic approximation to the marginal likelihood of that model, and it is equivalent to the minimum description length proposed by Rissanen (1987), and adopted to a decomposable consistent score for Bayesian networks by Lam and Bacchus (1994) and Friedman and Goldszmidt (1998). A Bayesian metric for scoring models was proposed by Cooper and Herskovits (1991) and generalized in (Cooper and Herskovits, 1992). Cooper and Herskovits (1992) also proposed a search algorithm (known

as the K2 algorithm) that performs a greedy search conditioned on a linear ordering of the variables (for literature on search in general, see (Michalewicz and Fogel, 2000)). Heckerman *et al.* (1995b) considered the specification of prior information such that equivalent network structures (Chickering, 1995) are given the same score. In the context of equivalent structures, greedy search procedures have been proposed by Chickering (2002); Chickering and Meek (2002) that are guaranteed to identify the correct structure when the amount of data grows large. (Chickering *et al.*, 2004) is one of the latest in a line of results that show that the task of learning Bayesian network structures is NP-hard, and Cowell (2001) has shown that, under often-quoted assumptions, constraint-based learning and score based learning are equivalent. In the context of missing data, Friedman (1998) has proposed a structural learning method that follows the intuition of the EM-algorithm. Finally, Friedman and Koller (2003) provide a method for calculating the posterior probability of absence or presence of individual arcs in the generating net given the data. Other sources of literature that can be recommended for further reading include (Buntine, 1996), (Heckerman, 1998), (Jordan, 1998), and (Cowell *et al.*, 1999).

7.6 Exercises

Exercise 7.1. Apply the PC algorithm to learn a skeleton over the six variables A, B, C, D, E, and F (use the network structure in Figure 7.23 as an oracle). Using rules 1–4, exploit the identified independence statements to set directions on the links in the skeleton.

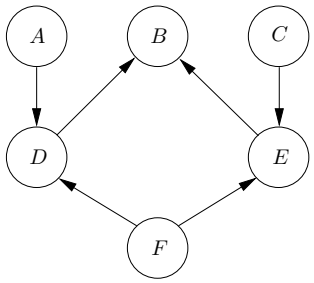


Fig. 7.23. Use the Bayesian network structure as an oracle for Exercise 7.1.

Exercise 7.2. Use rules 1–4 to set the directions on the remaining links in the structure in Figure 7.24.

Exercise 7.3. Assume that the PC-algorithm is run on five variables A, B, C, D, and E. During its running time, the algorithm gets positive

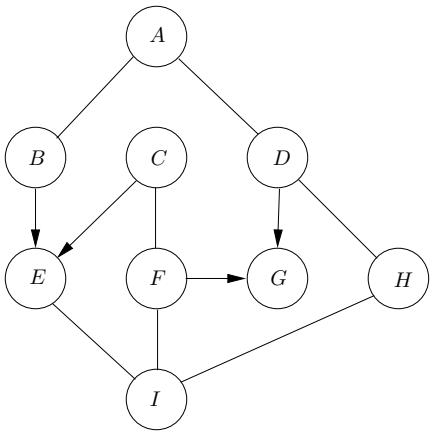


Fig. 7.24. A partial Bayesian network structure found by the PC algorithm and rule 1.

replies only to the following oracle queries: $I(A, B, E)$, $I(A, C, E)$, $I(A, D, E)$, $I(B, E, \{C, D\})$, and $I(C, D)$. The result of the run is a Bayesian network M. What does the skeleton of M look like? Which graphs can M be?

Exercise 7.4.

(i) Find a (tight) upper bound on the number of independence tests performed by the PC algorithm.
(ii) Discuss an implementation strategy for the PC algorithm with focus on the time used to perform the independence tests required by the algorithm (that is, calculating the conditional mutual independence expression, equation (7.2)).

Exercise 7.5. Prove that

$$I(A, B, \mathcal{X}) \Leftrightarrow \mathrm{CMI}(A, B|\mathcal{X}) = 0.$$

Exercise 7.6. Show that the size of the BN in Figure 7.13(c) is larger than the size of the BN in Figure 7.13(a).

Exercise 7.7. What is the size (see Proposition 7.1) of the BN shown in Figure 7.3(c) assuming that all variables are ternary.

Exercise 7.8. What is the BIC score, based on the data in Table 7.4, for the structure in Figure 7.25? What is the score for the structure in Figure 7.26?

Exercise 7.9. ^{E} Calculate the BIC score for the model of the simplified insemination problem described in Section 3.1.3, based on the (incomplete) database in Example 6.2.

C	B	A	C	B	A
1	1	1	1	1	2
1	1	1	1	1	2
1	1	2	2	2	1
2	1	2	1	1	2
1	1	1	2	1	2
1	1	2	1	1	1
2	1	2	1	1	1
1	1	1	2	1	2
1	1	1	1	1	2
1	1	2	1	1	1
1	1	1	1	1	1
1	1	1	2	1	2
1	1	2	1	1	2
2	2	1	1	1	2
1	1	2	2	2	1
2	1	2	1	1	2

Table 7.4. A number of configurations over binary variables A, B, and C.

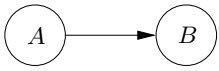


Fig. 7.25. A Bayesian network for Exercise 7.8.

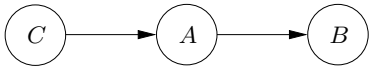


Fig. 7.26. A Bayesian network Exercise 7.8.

Exercise 7.10. What is the result of running greedy search based on the BIC score and the data in Table 7.4 starting from the empty graph?

Exercise 7.11. Show that the two expressions (in equation (7.4) and equation (7.5)) for the BIC score are identical.

Exercise 7.12. Tabu search is a general search technique based on greedy search. The technique tries to avoid getting stuck in local minima by prohibiting moves that involve aspects that were changed by a recent move. How could Algorithm 7.2 be modified to incorporate this behavior?

Exercise 7.13. Simulated annealing is a general search technique based on greedy search. The technique tries to explore more parts of the search space by making totally random moves at first, ignoring the score of the parts of the search space it moves to. Gradually it starts letting the scores influence the search, and finally ends up moving only if the score improves (as greedy search always behaves). How could Algorithm 7.2 be modified into a simulated

annealing search algorithm? (Hint: Use a counter i that is decreased at each iteration, and an error term like e^{-i} to modify scores.)

Exercise 7.14. What network structures are equivalent to the network in Figure 7.27?

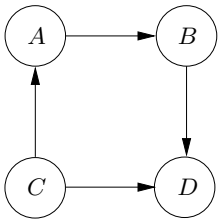


Fig. 7.27. A Bayesian network for Exercise 7.14.

Exercise 7.15. Learn a Chow–Liu tree from the data in Table 7.4.

Exercise 7.16. Complete Example 7.4 by calculating the BD score for the Bayesian network structure shown in Figure 7.28 based on the database in Table 7.2. As in Example 7.4 we assume that all network structures are a priori equally probable and that we have a uniform prior over all the possible parameters.

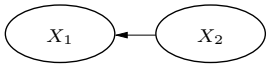


Fig. 7.28. Together with the network structures shown in Figure 7.14, this BN structure defines the space of model structures covering two variables.

(i) Calculate $P(\mathcal{D})$, the prior probability of the data.
(ii) Calculate the conditional probabilities for the three network structures given the database.
(iii) What should the prior probability for the empty graph (at least) have been for it to be picked by the BD score? Give an intuitive reason.

Exercise 7.17. Consider the database in Table 7.2 and a prior network structure consisting of an arc from X_2 to X_1. What is the result of learning with a greedy search and the BD score introduced in (7.7)?

Exercise 7.18. Show that when using a nonempty prior network structure together with equation (7.10), the resulting prior distribution cannot be score equivalent.