Learning Bayesian Network Model Structure from Data

Dimitris Margaritis

May 2003

CMU-CS-03-153

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Thesis Committee:
Sebastian Thrun, Chair
Christos Faloutsos
Andrew W. Moore
Peter Spirtes
Gregory F. Cooper (University of Pittsburgh)

This research was sponsored by the Defense Advanced Projects Agency (DARPA) and the US Air Force Research Laboratory (AFRL) under grant no. F30602-98-2-0137, the US Army under contract no. DAAE-07-98-CL-032, the National Science Foundation (NSF) under grant no. IIS-9877033 and no. SBR-9720374, and Draper Laboratories. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the sponsors or any other entity.

Keywords: Bayesian networks, Bayesian network structure learning, continuous variable independence test, Markov blanket, causal discovery, DataCube approximation, database count queries.

Abstract

In this thesis I address the important problem of the determination of the structure of directed statistical models, with the widely used class of Bayesian network models as a concrete vehicle of my ideas. The structure of a Bayesian network represents a set of conditional independence relations that hold in the domain. Learning the structure of the Bayesian network model that represents a domain can reveal insights into its underlying causal structure. Moreover, it can also be used for prediction of quantities that are difficult, expensive, or unethical to measure—such as the probability of lung cancer for example—based on other quantities that are easier to obtain. The contributions of this thesis include (a) an algorithm for determining the structure of a Bayesian network model from statistical independence statements; (b) a statistical independence test for continuous variables; and finally (c) a practical application of structure learning to a decision support problem, where a model learned from the database—most importantly its structure—is used in lieu of the database to yield fast approximate answers to count queries, surpassing in certain aspects other state-of-the-art approaches to the same problem.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my advisor, Sebastian Thrun, for his unwavering support throughout the duration of my time as a graduate student at CMU. His cheerfulness and encouragement have been essential in giving me the space that allowed me to discover my academic interests and passions and mature as a researcher.

I would also like to thank my friends and fellow graduate students at CMU, present and past, who have made life enjoyable: Paul Bennett, Joyoni Dey, Frank Dellaert, Joaquín López Fernández, Alex Gray, Kwun Han, Stavros Harizopoulos, Yannis Koutis, John Langford, Shyjan Mahamud, Michael Mateas, Kamal Nigam, Daniel Nikovski, Eugene Ng, Spiros Papadimitriou, Joelle Pineau, Jovan Popović, Chuck Rosenberg, and Nick Roy.

I owe special thanks to my committee members, Greg Cooper, Christos Faloutsos, Andrew Moore and Peter Spirtes, for their useful and constructive comments and advice.

Last but not least I would like to thank all the helpful, thoughtful and generally wonderful staff at the CMU Computer Science department, especially Sharon Burks and Catherine Copetas, that have made it such a nice place to spend all these years.

Chapter 1

Introduction and Motivation

1.1 Why Models?

How do people learn things that arguably involve intelligence such as how to drive a car? Among the first things one learns is that pressing the gas pedal makes the car move, and that rotating the steering wheel turns the wheels in the corresponding direction, which in turn determines the direction of the motion. These are examples of what are sometimes called *cause-effect* relationships. In this thesis, our focus is on learning this kind of relationships from observations of the environment, and in addition compiling them into a consistent mesh that may be used to describe the way the world, or at least an appropriate abstraction of it, works. We shall call such meshes or networks of cause-effect relationships *causal models*.

Since ancient times people have used models as a means of coping with the variability and the complexity of their environment. This is because a model can be used in situations other than the ones where it was learned. For example knowledge of how to drive your car, which may happen to be a sedan, can help you in driving your friend's station-wagon, even though the two tasks differ in the details. The reason you may be able to do this is because you are using a model of driving that is robust enough so that it can be applied to similar situations. In this capacity a model is in effect a "pattern" that may be reused as long as it matches sufficiently to the circumstances of the environment.

In another capacity, models can be used to hide complexity. In our description of the effects of turning the steering wheel for example, we are hiding the fact that in reality this action rotates a shaft that turns a crank arm that in turn converts the rotating motion of the shaft to a left-right movement of tie-rods that are attached to the front wheels, making them face the corresponding direction. This low-level description of the process can be repeated in increasing levels of detail, theoretically down to the elementary particles of the objects participating in the operation. Using a high-level model hides all this complexity by employing a simple cause-effect relationship involving the steering wheel and the car wheels only.

What is the usefulness of models today? Of course, a computer program for learning the rules for driving a car such as these may not be very useful to ordinary humans wanting to learn how to drive. But it may be useful in other situations where automating behavior is desirable, such as a robot operating machinery for example, helping in accelerating and simplifying its training. More importantly, and beyond this simple example, causal models can do much more for humans. They can help us understand our environment and discover "laws" of nature in the sciences: Biology, Genetics, Chemistry, even Physics. Practical systems in the same vein today can and do help doctors narrow the choices of their diagnosis for diseases. In that capacity, automatic or even semi-automatic construction of models can be invaluable.

1.2 Why *Bayesian Network* Models?

In this thesis we chose to focus on a special class of models called **Bayesian Networks (BNs)**. They are *graphical models*, which means that they contain a part that can be depicted as a graph. The reasons for our choice are multiple. First and foremost, we needed a concrete class of models on which to demonstrate and apply our ideas, and also on which to evaluate the resulting algorithms. Second, we wanted to use probability theory as our foundation, which is an old and tried theory that has withstood the test of time and has become one of the cornerstones of the sciences. Our use of probability theory comes from necessity: most AI application domains involve uncertainty, with which we need to deal with explicitly and from the start in a principled way. Finally, we are interested in deriving cause-effect relationships from data. Even though many classes of models can be used to represent uncertain domains—for example, decision trees, artificial neural networks, mixtures of basis functions, Markov networks *etc.*—only in the Bayesian network literature do we find claims of being able to represent and learn directed causal relationships, the discovery of which is our ultimate research goal.

In summary, the reasons for choosing Bayesian networks as a vehicle for our ideas are:

1. They are graphical models, capable of displaying relationships clearly and intuitively.
2. They are directional, thus being capable of representing cause-effect relationships.
3. They can handle uncertainty.
4. They handle uncertainty though the established theory of probability.
5. They can be used to represent indirect in addition to direct causation.

Below we briefly describe the dual nature of BN models, namely their ability to represent causal relationships and joint probability distributions.

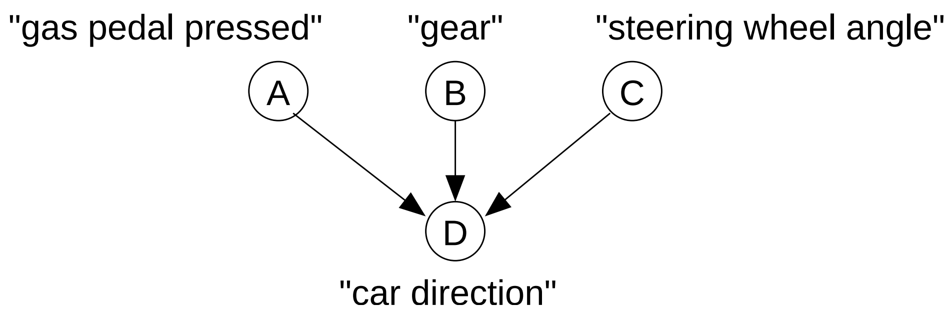


Figure 1.1: An example Bayesian network that can be used for modeling the direction of a car.

1.2.1 Causal Discovery

BNs correspond to a very broad class of models, one that can be used to represent nested, acyclic statistical models of virtually any kind of non-pathological joint probability distribution. Their signature characteristic is their ability to encode *directional* relations which can represent cause-effect relationships, compared to other graphical models that cannot *e.g.* Markov networks. As an added benefit, they are capable of representing many of the independencies in a domain through their structure, which is a directed acyclic graph. These two characteristics are intimately related: independencies are the direct effect of the causal relationships present, and the algorithms that are presented in this thesis and elsewhere rely on their presence. The reverse however is not true: the ability to represent independencies does not automatically lead to models that encode causal relationships. One such example is decision trees, which make no such guarantee. As we mentioned above, the ability to represent directional relationships is an important reason for choosing to focus on them in this thesis.

In this thesis we determine causal structure using constraints that are independence tests only. Other kinds of constraints can also be use to restrict the space of legal structures appropriately (Verma and Pearl, 1990); however we do not pursue these here. A more detailed discussion on the independencies that are implied by a BN structure can be found in section 2.3, and of algorithms that learn the structure from independencies in section 2.7.3.

To make our discussion of cause-effect relationships and independencies more clear and concrete, in Fig. 1.1 we depict a BN model that may be used for the example mentioned at the beginning of this chapter, to model the direction of a car at some high level of abstraction. According to the model in the figure, the direction of the motion of a car is directly caused by whether or not the gas pedal is pressed, what gear is shifted (forward or reverse), and the angle of the steering wheel. This is the causal model that a person might reasonably have in her mind in this toy domain. The connection of BN structure to independencies in the domain in this BN is that this model implies that in this domain the top three variables (namely “gas pedal pressed,” “gear” and “steering wheel angle”) are independent. Most algorithms for recovering the causal structure operate in the reverse direction, *i.e.* by examining the independence relations that can be observed in the domain and making causal conclusions, if possible.

Speaking of causal discovery however, we have to hint on caution: while many BN structure induction algorithms do give a guarantee on recovering causal relations, they are only able to do so but only under some rather substantial assumptions, and only when a special class of methods are used to construct them from data (constraint-based methods). The assumptions needed are described in section 2.6.

1.2.2 Succinct Probability Distribution Representation

In addition to their ability to represent causal relationships, BNs can and have been used to represent joint probability distributions (**pdf**s) *compactly*. Indeed this is the most common usage of them today. This ability comes from *local pdfs* that are attached to each variable in the network, whose purpose is to quantify the strength of the causal relationships depicted in the BN through its structure: these local pdfs mathematically describe the behavior of that variable under every possible value assignment of its parents. Since to specify this behavior one needs a number of parameters exponential in the number of parents,[1] and since this number is typically smaller than the number of variables in the domain, this results in exponential savings in space (for storing the parameters of the BN) and time (when using the BN to do estimation of functions of a subset of the variables in the domain).

More concretely, given the structure and the local pdfs of a BN, the joint pdf of the domain of n variables $\Pr(X_1, X_2, \ldots, X_n)$ can be calculated as

$$\Pr(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} \Pr(X_i \mid \mathbf{Pa}_i)$$

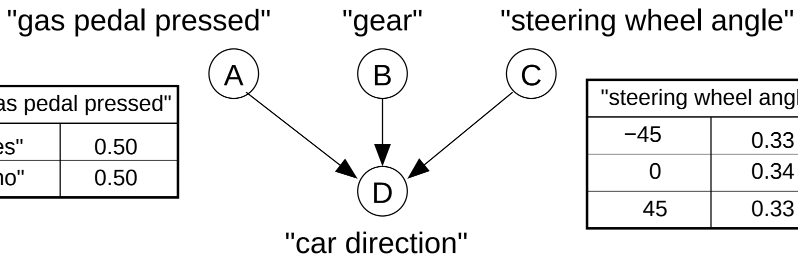
where $\mathbf{Pa}_i$ are the parents of variable X_i in the Bayesian network whose structure is $\mathcal{G}$. The conditional probabilities $\Pr(X_i \mid \mathbf{Pa}_i)$ defining the pdf of variable X_i given a value assignment of its parents $\mathbf{Pa}_i$ in the graph in this equation are exactly those local pdfs specified for each variable in the domain. The conditional probabilities $\Pr(X_i \mid \mathbf{Pa}_i)$ can be specified by $O(2^{|\mathbf{Pa}_i|+1})$ rather than $O(2^n)$ parameters, resulting in the exponential space savings mentioned above.

Using the above formula, every combination of assignments to the variables $X_1, \ldots, X_n$ can be calculated. We give an example of this calculation on the example from the previous section, using the conditional probability tables shown in Fig. 1.2. All variables in this domain are assumed discrete and the local pdfs are multinomial distributions (for convenience). According to the BN model of Fig. 1.2, the configuration ("gas pedal pressed" = "yes," "gear" = "forward," "steering wheel angle" = 45, "car direction" = 0) for example has probability

$$\Pr(A = \text{yes}, B = \text{forward}, C = 45, D = 0) \quad = \quad \Pr(D = 0 \mid A = \text{yes}, B = \text{forward}, C = 45)$$

[1] When the local pdfs are multinomial distributions. This is the most common choice for categorical variables.

"gear"	
"forward"	0.50
"backward"	0.50



"gas pedal pressed"	
"yes"	0.50
"no"	0.50

"steering wheel angle"	
−45	0.33
0	0.34
45	0.33

car direction	"gas pedal pressed" = "yes"						"gas pedal pressed" = "no"					
	"gear" = "forward"			"gear" = "backward"			"gear" = "forward"			"gear" = "backward"		
	"Steering wheel angle"											
	−45	0	45	−45	0	45	−45	0	45	−45	0	45
−60	0.40	0.00	0.00	0.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00
−30	0.50	0.25	0.00	0.00	0.25	0.50	0.00	0.00	0.00	0.00	0.00	0.00
0	0.10	0.50	0.10	0.10	0.50	0.10	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.25	0.50	0.50	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60	0.00	0.00	0.40	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 1.2: An example Bayesian network that can be used for modeling the direction of a car, together with the local conditional probability tables attached to each variable.

$$
\begin{aligned}
&\times \quad \Pr(A = \text{yes}, B = \text{forward}, C = 45) \\
&= \quad \Pr(D = 0 \mid A = \text{yes}, B = \text{forward}, C = 45) \\
&\times \quad \Pr(A = \text{yes}) \times \Pr(B = \text{forward}) \times \Pr(C = 45) \\
&= \quad 0.10 \times 0.50 \times 0.50 \times 0.33 \\
&= \quad 0.00825.
\end{aligned}
$$

In the above calculation we made use of the fact that, according to the structure of the BN, the top three variables A, B and C are unconditionally independent, and therefore their joint probability can be written as the product of their marginal probabilities *i.e.* $\Pr(A,B,C) = \Pr(A)\Pr(B)\Pr(C)$. The joint probability for any value assignment to all variables in the domain can be calculated in a similar fashion. In addition, partial assignments can also be calculated. This can be done in several ways. The simplest one is marginalization, which involves summing (or integrating for continuous variables) over those variables that do not appear in the expression whose probability we are interested. For example, the probability that the car direction is 0 degrees is

$$
\begin{aligned}
\Pr(D=0) &= \sum_{A\in\{\text{yes,no}\}} \sum_{B\in\{\text{forward,backward}\}} \sum_{C\in\{-45,0,45\}} \Pr(D=0 \mid A,B,C)\Pr(A,B,C) \\
&= \sum_{A\in\{\text{yes,no}\}} \sum_{B\in\{\text{forward,backward}\}} \sum_{C\in\{-45,0,45\}} \Pr(D=0 \mid A,B,C)\Pr(A)\Pr(B)\Pr(C) \\
&= 0.10\times 0.50\times 0.50\times 0.33 + 0.50\times 0.50\times 0.50\times 0.34 + 0.10\times 0.50\times 0.50\times 0.33 + \\
&\quad\; 0.00\times 0.50\times 0.50\times 0.33 + 0.00\times 0.50\times 0.50\times 0.34 + 0.00\times 0.50\times 0.50\times 0.33 \\
&= .05075.
\end{aligned}
$$

Note that all the entries in the sum can be read off the conditional probability tables of Fig. 1.2. The presence of independencies, such as the independence of A, B, and C, considerably speeds up the calculation of the joint, since it reduces the number of entries we need to sum over. This speed savings is typically exponential.

Beyond this simple example, a BN model can be used to calculate the probability of any event (conjunction of variable assignments) involving variables in the domain, conditional on any other event. This process is called *probabilistic inference*. In general probabilistic inference is NP-complete, because any algorithm might need to marginalize over an exponential number of variable assignments in the worst case, *i.e.* when many dependencies exist in the domain. However in many practical applications a significant number of independencies is present, making inference tractable.

There exist a number of algorithms whose purpose is to automate probabilistic inference. Such algorithms include exact methods such as cutset conditioning (Pearl, 2nd Ed., 1997; Darwiche, 1995; Suermondt and Cooper, 1990), junction trees (Lauritzen and Spiegelhalter, 1988; Jensen et al., 1990; Huang and Darwiche, 1994), node removal (Schachter, 1990) and symbolic manipulation (Chang and Fung, 1991; Schachter et al., 1990), as well as a number of approximate ones such as logic sampling (Henrion, 1988) and Gibbs sampling (Pearl, 1987; Chavez and Cooper, 1990).

1.3 Thesis Goals and Overview

The goal of this thesis is

> *to develop new techniques for recovering the structure of Bayesian network models from data, as well as demonstrate their utility by applying existing ones to interesting and challenging problems.*

The remainder of the thesis document is structured as follows:

In **Chapter 2** we describe the components of Bayesian networks models (structure and parameters) in detail, and give some background on learning these from data.

In **Chapter 3** we present a new algorithm on learning the structure of Bayesian networks from independence tests, by first mapping the neighborhood of each variable, using a new, simple and efficient algorithm.

In **Chapter 4** we present work toward developing the low-level tools for the algorithm of Chapter 3 (and other similarly inspired algorithms in the literature), namely a non-parametric statistical independence test to be used in learning the structure of Bayesian networks in continuous domains of arbitrary distribution.

Finally, in **Chapter 5** we demonstrate the usefulness of automatically learning the structure of Bayesian networks by solving a difficult problem from the database literature, namely approximate query answering for very large databases.

Chapter 2

Bayesian Network Models

In this chapter we describe in some detail certain aspects of Bayesian networks. The chapter is structured as follows. After a description of general notation to be used in section 2.1, we introduce BN models and give a simple example. The rules that govern the way direct and induced independencies expressed in a BN model are briefly addressed in Section 2.3, together with some examples. The formal definition of a BN is given in Section 2.4, followed by a discussion of the local pdfs and their use in Section 2.5. In Section 2.6 we describe our assumptions that are necessary for learning the causal structure of BNs using independence tests. In Section 2.7.1 we discuss learning the parameters of a BN once the structure is known. In Section 2.7.2 we describe approaches to learning the structure of BN models which are guided by a score function that measures how well the data are described by a candidate model. Finally, in Section 2.7.3 we present methods for learning the structure of BN models from constraints, which for the purposes of this thesis are probabilistic independencies.

2.1 Preliminaries: Notation

Table 2.1 contains the symbols that are used throughout the thesis. As noted, regular variables appear in capitals while capital bold-faced letters indicate sets. For example, $\mathcal{U}$ is the set of variables, which coincides with the set of nodes in the corresponding Bayesian network. The notation $X \not\perp Y \mid \mathbf{S}$ denotes that variables X and Y are dependent upon conditioning on (at least one value assignment of) the variables in the set $\mathbf{S}$, while $X \perp Y \mid \mathbf{S}$ indicates conditional independence.

We consider the terms "node," "variable," and "attribute" interchangeably throughout the thesis, and similarly for the terms "edge" and "arc."

Table of Symbols	
$\mathcal{D}$	Main data set
N	Number of points in data set *i.e.* $\lvert\mathcal{D}\rvert$
$X,Y,Z,\ldots$	One-dimensional variables
$x,y,z,\ldots$	Values of corresponding variables $X,Y,Z,\ldots$
$\mathbf{S},\mathbf{T},\ldots$	Sets
$\mathcal{U}$	Universe, set of variables / nodes in the domain: $\{X_1,\ldots,X_n\}$
n	Number of variables *i.e.* $\lvert\mathcal{U}\rvert$
$\mathcal{E}$	Set of edges of a BN
$\mathcal{T}$	Set of parameters of local pdfs for entire BN *i.e.* $p_{ijk}, i=1,\ldots,n, j=1,\ldots,q_i, k=1,\ldots,r_i$
m	Number of edges of the BN *i.e.* $\lvert\mathcal{E}\rvert$
$\mathcal{G}$	Directed acyclic graph (DAG) of a BN *i.e.* $\langle\mathcal{U},\mathcal{E}\rangle$
$\mathcal{B}$	Bayesian network, consists of DAG and parameters *i.e.* $\langle\mathcal{G},\mathcal{T}\rangle=\langle\mathcal{U},\mathcal{E},\mathcal{T}\rangle$
$\mathbf{B}(X)$	Markov blanket of variable X
$\mathbf{N}(X)$	Set of direct neighbors of variable X in Bayesian network
$\mathbf{Pa}_i$	Set of parents of X_i
$\mathbf{pa}_{ij}$	Set of values for value assignment j of each member of the set of parents $\mathbf{Pa}_i$ of X_i
r_i	Number of values of discrete variable X_i
q_i	Number of configurations of the set of parents of X_i
$c_1,c_2,\ldots,c_K$	Counts of a multinomial distribution with K bins
$p_1,p_2,\ldots,p_K$	Parameters (bin probabilities) of a multinomial distribution with K bins
$\alpha_i,\beta_j,\gamma_k$	Hyperparameters of multinomial distributions

Table 2.1: Symbols used throughout the thesis.

2.2 Bayesian Network Models

A Bayesian network is a graphical representation of a probability distribution over a set of variables $\mathcal{U}=\{X_1,X_2,\ldots,X_n\}$. It consists of two parts:

(a) the directed network structure in the form of a directed acyclic graph (**DAG**), and

(b) a set of the local probability distributions (**local pdfs**), one for each node/variable, conditional on each value combination of the parents.

The network structure is constrained to be acyclic. Undirected cycles are allowed *i.e.* cycles along which not all edges are pointed in the same way. Such structures represent alternative paths of possible influence between certain variables in the cycle.

A simple example of a Bayesian network in a discrete domain is shown in Fig. 2.1. It depicts a fictitious situation at a university campus, a domain abstracted to five variables, all of them boolean: A (representing

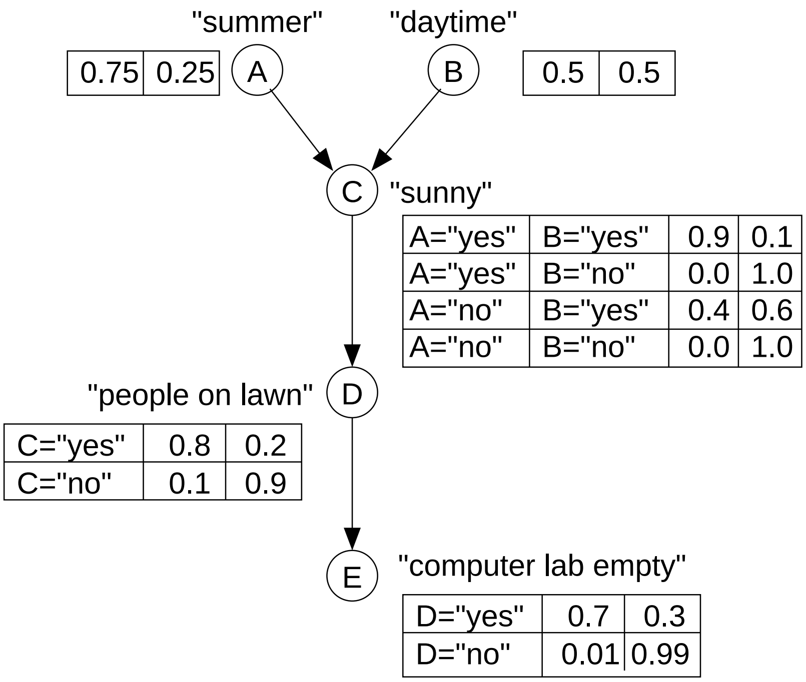


Figure 2.1: An example Bayesian network modeling the weather and computer lab activity.

the event that it is summer), *B* (representing the event that it is daytime), *C* (representing the event that it is sunny), *D* (representing the event that there are people lying on the lawn outside), and *E* (representing whether the computer lab is empty of people). In the domain of Fig. 2.1 where all variables are binary, each row of each conditional probability table records the probabilities of that variable taking the value "true" or "false" for a particular combination of values ("true" or "false") of its parents. For example, given that it is "summer" and "daytime", the probability that it is "sunny" is 0.9.

The intuitive meaning of the structure BN in Fig. 2.1 is that *A* depends on *B* and *C* but *B* and *C* are independent. Another statement implied by the BN is that *A* and *D* become independent once we know (fix) the value of *C*. In general, the meaning of a BN consists of a set of statistical conditional independence statements that are implied by its structure. Under certain assumptions (described in section 2.6), it also includes statements of dependence among variables. From a set of axioms described in Pearl (2nd Ed., 1997) and certain assumptions described later in section 2.6, one can produce the entire set of independence relations that are implied by that BN model. However, determining the independence relations that are entailed by $\mathcal{G}$ from these axioms can be cumbersome, because it requires their repeated use until the desired relation is proved or disproved. An equivalent approach is to "read" those independencies from the structure of a BN model using the rules of d-separation. Since this is significantly easier than using the set of axioms, it is frequently the approach of choice in practice. D-separation is an important set of rules invoked frequently through out this thesis. It is described in a separate section below.

2.3 D-separation

As mentioned above, the rules of d-separation can be used to "read" independencies that hold true in the domain given the structure $\mathcal{G}$ of a BN. The rules resemble graph connectivity, with some important differences. For example, in contrast to regular graph connectivity concepts, conditioning on a node may "block" or "unblock" a path of dependence between two nodes, depending on the direction of traversal of that node along that path. Let us assume that a node Z lies along an undirected path p between X and Y, and that p is the only path between X and Y in the graph $\mathcal{G}$ (for simplicity). We would like to determine whether X is independent of Y given Z. According to d-separation, Z is blocked if p traverses it along the following directions:

- coming from a child of Z, or
- coming from a parent of Z and exiting from a child of Z.

In the remaining case, namely if p traverses Z coming from a parent of Z and exiting from another parent of Z (assuming Z has at least two parents), Z is considered unblocked when conditioned on. When Z is not conditioned on, in all these cases, its status (blocked or unblocked) is inverted by removing the conditioning on it or any of its descendants.

These are the rules governing conditioning on one variable. When conditioning on more than one variable on a single path between two nodes X and Y, X and Y are independent iff there exists at least one node blocked along that path. More generally, there may be more than one undirected paths between X and Y in the graph. In that case X is independent of Y when all those paths are blocked.

The formal definition of d-separation, adapted from Pearl (1995) is:

Definition (d-separation): Let $\mathbf{S}$, $\mathbf{T}$, and $\mathbf{V}$ be three disjoint subsets of nodes in a DAG $\mathcal{G}$, and let p be any path between a node in $\mathbf{S}$ and a node in $\mathbf{T}$, where by a path we mean any succession of arcs, regardless of their directions. Then $\mathbf{V}$ is said to block p if there is a node Z on p satisfying one of the following two conditions:

1. Z has converging arrows (along p) and neither Z nor any of its descendants are in $\mathbf{V}$, or
2. Z does not have converging arrows (along p) and Z is in $\mathbf{V}$.

Furthermore, $\mathbf{V}$ is said to d-separate $\mathbf{S}$ from $\mathbf{T}$, written $(\mathbf{S} \perp_{\mathcal{G}} \mathbf{T} \mid \mathbf{V})$ iff $\mathbf{V}$ blocks every path from a node in $\mathbf{S}$ to a node in $\mathbf{T}$.

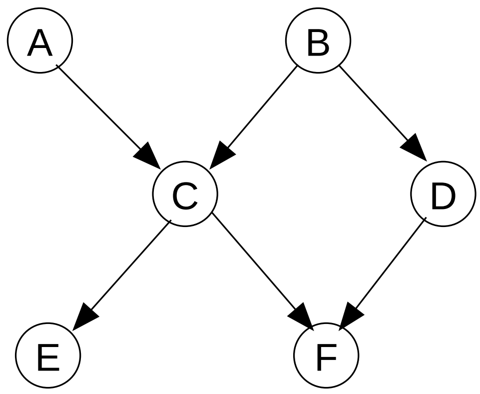


Figure 2.2: Example BN used to demonstrate d-separation.

In order to illustrate the rules of d-separation, we consider a simple example. Let us assume that the BN structure is as depicted in Fig. 2.2. Here are some examples of relations that hold true:

- $A \perp D$ (unconditional test) because both paths between A and D are blocked: path $A-C-B-D$ is blocked because C is blocked (neither C nor any of its descendants are conditioned on) and path $A-C-F-D$ is also blocked because F is blocked (not conditioned on).
- $A \not\perp D \mid C$ (under Faithfulness) because both C and B are unblocked: C is conditioned on and B is not.
- $A \not\perp D \mid E$ (under Faithfulness) because both C and B are unblocked: a descendant of C, namely F, is conditioned on and B is not.
- $A \perp D \mid \{B,C\}$, because even though C is unblocked, B is now blocked.
- $E \not\perp D$ (under Faithfulness), because both C and B are unblocked along the path $E-C-B-D$. Note that the alternative path, $E-C-F-D$ is blocked because we do not condition on F and therefore it is blocked.
- $E \perp D \mid B$, because now B is blocked.
- $E \not\perp D \mid \{B,F\}$, because now F is unblocked.

For details on d-separation, see Pearl (1995).

2.4 Bayesian Network Structure

The main purpose of the structure of a BN is to summarize a number of conditional independence relations, graphically. The formal definition of a BN is that of an **I-map** (Pearl, 2nd Ed., 1997):

Definition (Bayesian network): A DAG $\mathcal{G}$ is called an **I-map** of a probability distribution P if every conditional independence displayed on $\mathcal{G}$ through the rules of d-separation, are valid in P. More simply, for every $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$,

$$(\mathbf{X} \perp_{\mathcal{G}} \mathbf{Y} \mid \mathbf{Z}) \Rightarrow (\mathbf{X} \perp_{P} \mathbf{Y} \mid \mathbf{Z})$$

where $(\mathbf{X} \perp_{\mathcal{G}} \mathbf{Y} \mid \mathbf{Z})$ indicates d-separation of $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$, and $(\mathbf{X} \perp_{P} \mathbf{Y} \mid \mathbf{Z})$ denotes conditional independence in the underlying domain distribution P. $\mathcal{G}$ is a Bayesian network iff it is a minimal I-map of P, *i.e.* no edges can be removed from $\mathcal{G}$ without negating the I-map property.

In simple words,the above definition says that the structure implies a set of conditional independence relations among the variables involved. All of them are required to be valid. It is interesting to note that the implication is not required to go both ways; nodes d-connected (*i.e.* not d-separated) in the graph $\mathcal{G}$ are not necessarily dependent in the underlying distribution P. Therefore a completely connected DAG for example is always an I-map (implies no independencies) although perhaps not a valid BN according to the definition (if it is not minimal).

Besides independencies, the graph structure of a BN can also be used in certain domains to represent cause-effect relationships through the edges and their directions. In these cases, the parents of a node are taken to be the "direct causes" of the quantity represented by that node. However, that is only true under certain assumptions, the most important being the following two:

1. Whether or not there are any **common unobserved** causes (variables) of two or more observed variables in the domain. If there are no such variables, the property of causal sufficiency is said to hold true. Unobserved variables are also called **latent** or **hidden** variables.

2. Given causal sufficiency, whether or not it is possible for more than one network structure to fit the constraints that have been observed in the domain. These constraints are statistical independencies that are observed in the data for the purposes of this thesis. Only one of these networks can be the "true" underlying generative model that embodies the real cause-effect relationships that govern the data-generating mechanisms of the domain.

BNs can be used as causal models under the usual "cause-effect" interpretation when we can safely assume the above assumptions (which is rare, especially for the causal sufficiency). There is currently a great deal of interest and research toward detecting causal sufficiency *e.g.* using instrumental variables (Bowden and Turkington, 1984).

2.5 Bayesian Network Local pdfs

The second component of a BN is a set of local conditional probability distributions. Together with the graph structure, they are sufficient to represent the joint probability distribution of the domain. More concretely,

$$\Pr(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} \Pr(X_i \mid \mathbf{Pa}_i).$$

where $\mathbf{Pa}_i$ is the set containing the parents of X_i in the BN. In other words, the joint pdf of the domain can be *factorized* into smaller, local pdfs each involving a node and its parents only. Viewed in this way, the local pdfs provide the *quantitative* probabilities that, when multiplied together in the fashion prescribed by the *qualitative* independencies that are implied by the structure of the BN, are sufficient to reconstruct the joint pdf of the domain.

Any probability distribution family can be used for the local pdfs. The independencies displayed in the structure of the BN hold true for every member of the family that is consistent with the structure. In other words, they are true for any choice of parameters for the local pdfs. In practice, when a variable and its parent in the graph are discrete, these local pdfs are frequently represented by a multinomial distribution. When they are continuous, mixtures of Gaussians (Davies and Moore, 2000) and artificial neural networks (Monti and Cooper, 1998a) have been used in practice.

2.6 Assumptions for Learning the Causal Structure

As we mentioned in Section 2.4, a BN model encodes a set of independencies that exist in the domain. The existence of these independencies in the actual population depends on the extent to which these assumptions hold. These are:

Causal Sufficiency Assumption: There exist no common unobserved (also known as hidden or latent) variables in the domain that are parent of one or more observed variables of the domain.

Markov Assumption: Given a Bayesian network model B, any variable is independent of all its non-descendants in B, given its parents.

The Causal Sufficiency Assumption states that there are no unobserved variables in the domain that might explain the independencies that are observed in the data, or lack thereof. It is a crucial assumption for applications that need to determine the true underlying (causal) structure of the domain. Unfortunately it is the one most likely to be invalid in all but the most trivial domains. That is because it is usually easy to imagine one more variable, perhaps at a different level of detail that can be included in the model as an endogenous or even exogenous variable.[1] One such example is mentioned in the introduction (Section 1.1). There exists however promising ongoing research toward tests for determining the existence or not of latent variables in restricted situations, such as instrumental variables (Bowden and Turkington, 1984).

The Markov Assumption expresses a minimum set of independence relations that exist between every node and its non-descendants, given a BN model. From these, and a set of axioms described in Pearl (2nd Ed., 1997), one can produce the entire set of independence relations that are implied by that BN model. However, more than these may be valid in the probability distribution of the population. In other words, whenever an edge or an unblocked path exists between two nodes, it is not necessarily the case that these nodes are dependent. If they are, then that graph and distribution are said to be **faithful** to one another (Spirtes et al., 1993) or that the graph is a **D-map** of the distribution (Pearl, 2nd Ed., 1997). In other words, not only the independencies displayed in the graph are valid in the distribution, but also the dependencies (the lack of independencies). Faithfulness is an assumption usually made, and one we also assume to be valid throughout the thesis:

Faithfulness Assumption: A BN graph $\mathcal{G}$ and a probability distribution P are faithful to one another iff every one and all independence relations valid in P are those entailed by the Markov assumption on $\mathcal{G}$.

2.7 Learning Bayesian Networks

Perhaps the most challenging task in dealing with Bayesian networks is learning their structure. However, research in this direction is essential because of its enormous usefulness, as much for end-user applications (see for example Chapter 5) as for the learning of causal networks in Biology, Medicine, Chemistry, Physics, and in the sciences in general.

In this section we shall present an overview of the prevalent techniques that are used for learning Bayesian networks. In Section 2.7.1 we first describe how the parameters of BNs can be learned, given the structure. In the subsequent sections we focus on learning the structure itself. There are two broad classes of algorithms for learning the structure of BNs. One class "scores" a BN based on how well it fits the data, and attempts to produce one that optimizes that score. This class of algorithms is presented in Section 2.7.2. In Section 2.7.3 immediately following, we present the alternative approach, which uses constraints such as independence relations that we may know exist in the data, to reconstruct the structure. The latter class is more relevant to this thesis, although we do apply score-based learning to an application where fitting data well is a priority in Chapter 5.

[1] Endogenous variables are those with at least one parent. Exogenous have no observed variables as parents, although they can have one or more observed variables as children.

2.7.1 Learning the Parameters

Learning the parameters given a fixed network structure (statistical model) is a well-known problem in statistics. In the BN literature, where Bayesian approaches seem to be dominant, the problem is posed as follows. A prior distribution is assumed over the parameters of the local pdfs before the data are used (for example, this can be uniform). The conjugacy of this prior distribution is desirable; a distribution family is called **conjugate prior** to a data distribution when the posterior over the parameters belongs to the same family as the prior, albeit with different **hyperparameters** (the parameters of a distribution over parameters are sometimes called hyperparameters). In this thesis we use multinomials for the local pdfs only and therefore we shall present only this case in some detail here. For other cases, such as linear regression with Gaussian noise, see Buntine (1993); Heckerman and Geiger (1995), or for more complicated ones representable by artificial neural networks see Monti and Cooper (1998a).

For multinomial distributions the conjugate prior comes from the Dirichlet family. Denoting the probability of each bin $p_{ijk}, k = 1, \ldots, r_i$ in the local pdf of variable X_i for the parent configuration $\mathbf{pa}_{ij}$, the Dirichlet distribution over these parameters is expressed by[2]

$$\Pr(p_{ij1}, p_{ij2}, \ldots, p_{ijr_i} \mid \mathcal{G}) = \mathrm{Dir}(\alpha_{ij1}, \alpha_{ij2}, \ldots, \alpha_{ijr_i}) = \Gamma(\alpha_{ij}) \prod_{k=1}^{r_i} \frac{p_{ijk}^{\alpha_{ijk}-1}}{\Gamma(\alpha_{ijk})}$$

where α_{ijk} are its hyperparameters and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$. Assuming **local** and **global parameter independence** (Spiegelhalter and Lauritzen, 1990; Cooper and Herskovits, 1992; Heckerman et al., 1995a), the distribution over the set of parameters $\mathbf{p}$ of the entire BN is

$$\Pr(\mathbf{p} \mid \mathcal{G}) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \Gamma(\alpha_{ij}) \prod_{k=1}^{r_i} \frac{p_{ijk}^{\alpha_{ijk}-1}}{\Gamma(\alpha_{ijk})}.$$

Conditional on the data set $\mathcal{D}$, the posterior probability over the parameters is also a member of the Dirichlet family, since it is conjugate prior to the multinomial. It is

$$\Pr(p_{ij1}, p_{ij2}, \ldots, p_{ijr_i} \mid \mathcal{G}, \mathcal{D}) = \mathrm{Dir}(N_{ij1} + \alpha_{ij1}, N_{ij2} + \alpha_{ij2}, \ldots, N_{ijr_i} + \alpha_{ijr_i})$$

and

$$\Pr(\mathbf{p} \mid \mathcal{G}, \mathcal{D}) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \Gamma(\alpha_{ij} + N_{ij}) \prod_{k=1}^{r_i} \frac{p_k^{N_{ijk}+\alpha_{ijk}-1}}{\Gamma(N_{ijk} + \alpha_{ijk})},$$

where N_{ijk} is the number of samples in the bin k of the pdf for X_i for parent configuration $\mathbf{pa}_{ij}$ (N_{ijk} are the sufficient statistics of that pdf). Using this distribution to predict the value of any quantity $Q(X_1, X_2, \ldots, X_n)$ depending on the variables of the domain, one averages over all possible values of the (unknown) parameters,

[2] The gamma function is defined as $\Gamma(x) = \int_0^{+\infty} e^{-t} t^{x-1} \mathrm{d}t$. For the case where x is a non-negative integer, $\Gamma(x+1) = x!$.

weighted by the posterior probability of each value:

$$\Pr(Q(X_1,X_2,\ldots,X_n) \mid \mathcal{G},\mathcal{D}) = \int Q(X_1,X_2,\ldots,X_n)\Pr(\mathbf{p} \mid \mathcal{G},\mathcal{D})\mathrm{d}\mathbf{p}.$$

More often, for reasons of convenience, the maximum likelihood (ML) parameters are used instead of the entire distribution. The ML for p_{ijk} is

$$\hat{p}_{ijk} = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}}.$$

Also for convenience, especially in cases where data are abundant, the hyperparameters are ignored (assuming $\alpha_{ijk} \ll N_{ijk}$), and the fraction N_{ijk}/N_{ij} is used instead. This happens for example in score-based methods (see Section 2.7.2), where the BIC score is employed, which is itself a large-sample approximation of the posterior and is already assuming that the effects of a prior are negligible.

2.7.2 Learning the Structure: Score-based Methods

One of the most popular methods of inducing BNs from data, especially for the purpose of pdf estimation, is the score-based approach. The process assigns a score to each candidate BN, typically one that measures how well that BN describes the data set $\mathcal{D}$. Assuming a structure $\mathcal{G}$, its score is

$$Score(\mathcal{G},\mathcal{D}) = \Pr(\mathcal{G} \mid \mathcal{D}),$$

in other words, the posterior probability of $\mathcal{G}$ given the data set. A score-based algorithm attempts to maximize this score. Computation of the above can be cast into a more convenient from by using Bayes' law:

$$Score(\mathcal{G},\mathcal{D}) = \Pr(\mathcal{G} \mid \mathcal{D}) = \frac{\Pr(\mathcal{D} \mid \mathcal{G})\Pr(\mathcal{G})}{\Pr(\mathcal{D})}.$$

To maximize this we need only maximize the numerator, since the denominator does not depend on $\mathcal{G}$. There are several ways to assess $\Pr(\mathcal{G})$ from prior information, see Heckerman (1995) for a discussion and pointers to the literature. For the purposes of this discussion we will ignore $\Pr(\mathcal{G})$, which is equivalent to assuming a uniform prior over structures.

To calculate $\Pr(\mathcal{D} \mid \mathcal{G})$, the Bayesian approach averages over all possible parameters, weighing each by their posterior probability:

$$\Pr(\mathcal{D} \mid \mathcal{G}) = \int \Pr(\mathcal{D} \mid \mathcal{G},\mathbf{p})\Pr(\mathbf{p} \mid \mathcal{G})\,\mathrm{d}\mathbf{p}.$$

Cooper and Herskovits (1992) first showed that for multinomial local pdfs this is

$$\Pr(\mathcal{D} \mid \mathcal{G}) = \prod_{i=1}^{n}\prod_{j=1}^{q_{ij}} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

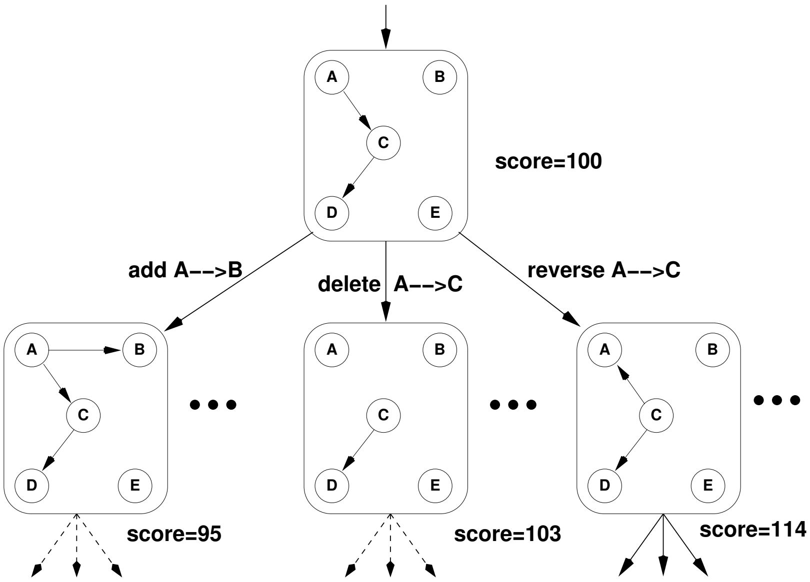


Figure 2.3: Illustration of a BN structure hill-climbing search procedure.

where α_{ijk} and N_{ijk} are the hyperparameters and counts for the pdf of X_i for parent configuration j. In the large sample limit the term $\Pr(\mathcal{D} \mid \mathcal{G}, \mathbf{p}) \Pr(\mathbf{p} \mid \mathcal{G})$ can be reasonably approximated as a multivariate Gaussian (Kass et al., 1988; Kass and Raftery, 1995). Doing that and in addition approximating the mean of the Gaussian with the maximum-likelihood value $\hat{\mathbf{p}}$ and ignoring terms that do not depend of the data set size N, we end up with the **BIC score** approximation:

$$BICscore(\mathcal{G}, \mathcal{D}) = \log \Pr(\mathcal{D} \mid \hat{\mathbf{p}}, \mathcal{G}) - \frac{d}{2} \log N, \tag{2.1}$$

first derived by Schwartz (1978). The term $\hat{\mathbf{p}}$ is the set of maximum-likelihood estimates of the parameters $\mathbf{p}$ of the BN, while d is the number of free parameters of the multivariate Gaussian, *i.e.* its number of dimensions, which coincides with the number of free parameters of the multinomial local pdfs *i.e.* $d = \sum_{i=1}^{n} q_i(r_i - 1)$. The usefulness of the BIC score comes from the fact that it does not depend on the prior over the parameters, which makes popular in practice in cases where prior information is not available or is difficult to obtain. It also has the intuitive interpretation of the data likelihood minus a "penalty term" ($-\frac{d}{2} \log N$) which has the effect of discouraging overly complicated structures and acting to automatically protect from overfitting. The BIC score has been shown to be equal to minus the MDL (Minimum Description Length) score (described by Rissanen (1987)).

As we mentioned above, score-based algorithms attempt to optimize this score, returning the structure $\mathcal{G}$ that maximizes it. This poses considerable problems since the space of all possible structures is at least exponential in the number of variables n: there are $n(n-1)/2$ possible undirected edges and $2^{n(n-1)/2}$

```
Procedure 𝓑 = BIChillclimb(𝓓)

1.  𝓔 ← ∅
2.  𝓣 ← ProbabilityTables(E, 𝓓)
3.  𝓑 ← ⟨𝓤, 𝓔, 𝓣⟩
4.  score ← −∞
5.  do:
    (a)  maxscore ← score
    (b)  for each attribute pair (X, Y) do
    (c)      for each 𝓔′ ∈ {𝓔 ∪ {X → Y},
                          𝓔 − {X → Y},
                          𝓔 − {X → Y} ∪ {Y → X}}
    (d)         𝓣′ ← ProbabilityTables(𝓔′, 𝓓)
    (e)         𝓑′ ← ⟨𝓤, 𝓔′, 𝓣′⟩
    (f)         newscore ← BICscore(𝓑′, 𝓓)
    (g)         if newscore > score then
                   𝓑 ← 𝓑′
                   score ← newscore
6.  while score > maxscore
7.  Return 𝓑
```

Figure 2.4: Pseudocode for the algorithm that constructs a BN from a data set $\mathcal{D}$ using hill-climbing search.

possible structures for every subset of these edges. Moreover, there may be more than one orientation of the edges for each such choice. Therefore a brute force approach that computes the score of every BN structure is out of the question in all but the most trivial domains, and instead heuristic search algorithms are employed in practice. One popular choice is hill-climbing, shown graphically in an example in Fig. 2.3 and in pseudocode in Fig. 2.4. The search is started from either an empty, full, or possibly random network, although if there exists background knowledge it can be used to seed the initial candidate network. The procedure *ProbabilityTables*() estimates the parameters of the local pdfs given a BN structure. Typically this is a maximum-likelihood estimation of the probability entries from the data set, which for multinomial local pdfs consists of counting the number of tuples that fall into each table entry of each multinomial probability table in the BN. The algorithm's main loop consists of attempting every possible *single-edge* addition, removal, or reversal, making the network that increases the score the most the current candidate, and iterating. The process stops when there is no single-edge change that increases the score. There is no guarantee that this algorithm will settle at a global maximum so a simple perturbation can be used to increase the chances of reaching a global maximum, multiple restarts from random points (initial networks) in the space, or simulated annealing can be used.

It is worthwhile to note that the restricted case of tree-structured BNs has been solved optimally, in the minimum KL-divergence sense, by Chow and Liu (1968). A similar approach has been proposed for the case of polytrees (trees where each node can have more than one parents) by Rebane and Pearl (1989), although its optimality is not proved.

Hill-climbing is not the only method of heuristic search. Best-first search (*e.g.* Russell and Norvig (1995)), genetic algorithms (Larranaga et al., 1996), and almost any kind of search procedure can also be used. A more principled approach is to reduce the search space by searching among independence-equivalent classes of networks instead (Chickering, 1996). Recently Chickering (2002) proved a conjecture of Meek (1997) that in the limit of large sample sizes, his GES (Greedy Equivalence Search) algorithm does identify an inclusion-optimal equivalence class of BNs *i.e.* a class of models such that (a) includes the probability distribution from which the dataset was drawn, and (b) no sub-model contains that distribution, if one exists. This is a significant result, although some open questions about the optimality of the resulting models remain when the distribution over the observed variables does not obey the composition property (see Pearl (2nd Ed., 1997) for details).

2.7.3 Learning the Structure: Constraint-based Methods

Using constraints is another way of learning BN structure. These constraints are typically conditional independence statements, although non-independence based constraints may be entailed by the structure, in certain cases where latent variables exist (*e.g.* see Verma and Pearl (1990)). However, since we are only concerned with domains without latent variables or missing values in this thesis, we will not refer to these any further.

The conditional independence tests that are used in practice are statistical tests on the data set. In order to use the results to reconstruct the structure, several assumptions have to be made: Causal Sufficiency, Causal Markov, and Faithfulness (see Section 2.6). With these assumptions in place, one can ascertain the existence of an edge between two variables, or the direction of that link, though the latter is only being possible in certain cases (see below).

The most straightforward algorithm proposed is the SGS (Spirtes et al., 1993), shown in Fig. 2.5. In that the existence of an edge between two variables, say X and Y, is tested using a number of conditional tests. Each of these conditions on a different subset of $\mathcal{U} - \{X,Y\}$. If Faithfulness holds and there exists an edge $X - Y$, then all these independence tests should be false. If there is no edge $X - Y$, then there must exist a subset d-separating them. Assuming that there is no direct edge between X and Y in the true model, one such subset is the set of parents of one of the nodes. By trying all possible subsets of $\mathcal{U} - \{X,Y\}$, the SGS algorithm can make a conclusion at the end of Step 1 on the existence of an edge between every pair of variables in the domain. After the undirected connectivity is determined, SGS attempts to determine the directionality of these edges. This is done by examining triples of variables X, Y, and Z, such that there

```
0.  ℰ ← ∅
1.  From a complete undirected graph with edge set ℰ' on the set of nodes 𝒰.
    For each pair of nodes X, Y, do:
       If for all subset S of 𝒰 − {X,Y}, X ⊥ Y | S,
          set ℰ' ← ℰ' − {(X,Y), (Y,X)}
2.  For each triple of nodes X, Y, Z, such that (X,Z) ∈ ℰ', (Y,Z) ∈ ℰ', (X,Z) ∉ ℰ', do:
       If for all subsets S of 𝒰 − {X,Y,Z}, X ⊥̸ Y | S ∪ {Z},
          set ℰ ← ℰ ∪ {(X,Z), (Y,Z)}
3.     Repeat:
          If (X,Y) ∈ ℰ, (Y,Z) ∈ ℰ', (X,Z) ∉ ℰ' and ∄W such that (W,Y) ∈ ℰ,
             set ℰ ← ℰ ∪ {(Y,Z)}
          If (X,Y) ∈ ℰ' and there is a directed path from X to Y in ℰ,
             set ℰ ← ℰ ∪ {(X,Y)}
       Until no more edges can be oriented (added to ℰ)
4.  Return Bayesian network 𝒢 with edge set ℰ.
```

Figure 2.5: Pseudocode for the SGS algorithm.

are $X-Z$ and $Y-Z$ edges but no $X-Y$ edge. If $X \not\perp Y \mid \mathbf{S}$, for all $\mathbf{S} = \{Z\} \cup \mathbf{S}', \mathbf{S}' \subseteq \mathcal{U} - \{X,Y,Z\}$, meaning no subset that includes Z can d-separate X and Y, then the directionality of $X-Z$ and $Y-Z$ is $X \to Y$ and $Y \to Z$, respectively. This is repeated for all such triples in Step 2, and is followed by Step 3 where the directions are propagated while maintaining acyclicity. As we mentioned above, the algorithm—and any other independence-based algorithm for that matter—cannot necessarily assign directions to every every edge. Doing so depends on the true underlying structure of the BN model. For example for a BN of three variables, $X \to Y \to Z$, the direction of either edge cannot be determined by any set of independence statements, because two other networks with the same undirected structure, namely $X \leftarrow Y \leftarrow Z$ and $X \leftarrow Y \to Z$, belong to the same equivalence class with respect to conditional independence statements implied by their respective structures.

Another algorithm, similar in flavor to the SGS is the IC, or "inductive causation" algorithm by Pearl and Verma (1991). Other algorithms exist in the literature that do not make use of independence tests but take into account d-separation in order to discover structure from data. Cheng et al. (1997) for example uses mutual information instead of conditional independence tests. The algorithm requires the ordering of the variables to be given to the algorithm in advance.

Constraint-based algorithms have certain disadvantages. The most important one, manifesting frequently in practice, is their poor robustness. By the term "robustness" here we mean large effects on the output of the algorithm *i.e.* the structure of the BN, for small changes of the input *i.e.* single errors in the independence tests. The problem seems to have its roots on the dependency of later parts of the algorithms to earlier ones, something that seems difficult to avoid. For example, in the pseudocode of Fig. 2.5, step 2 determines the direction of pairs of edges that were found in step 1. Therefore a missing edge might prevent another edge's direction to be recovered (*e.g.* if the network was $X \to Z \leftarrow Y$ and the edge $X \to Y$ was missed in step 1, the direction of edge $Y \to Z$ cannot be recovered by steps 2 or 3). In addition, step 3 propagates the directions of edges determined in step 2, so an error in step 2 might be propagated in step 3, possibly resulting in a structure with directed cycles which is illegal under the present BN formulation.

Another disadvantage is that the SGS (and the IC) algorithm, as we can see from its algorithmic description, is exponential in time in the number of variables of the domain, because it is conducting independent tests conditional on all $2^{|\mathbf{S}|}$ subsets of set $\mathbf{S}$ with $\mathbf{S}$ containing $n-2$ variables. This makes it impractical for large domains of tens or even hundreds of variables. A more efficient algorithm, not described here, is the PC algorithm. Its efficiency comes from ordering the conditional independence tests, from small to large. The algorithm is presented in detail in Spirtes et al. (1993).

In Chapter 3 we present an independence-based algorithm called the GS algorithm. Its running time is better than the PC algorithm, although still exponential in the worst case. However its main advantage is the use of an intuitive concept called the Markov Blanket of a variable (the concept of the Markov Blanket was introduced by Pearl and is explained in Chapter 3), which makes it easier to evaluate in a semi-automated context where prior information is available by an expert who can verify the results of the independence tests involved. In addition, it provides a heuristic in cases where there might exist errors in the independence tests.

Chapter 3

Learning BN Structure using Independence Tests

Knowledge about the independencies that exist in a domain is an extremely useful piece of information that a research scientist can—and must—elicit early on during her investigation. The reason for this is the fact that conditional independence statements concerning observed quantities in the domain can reduce the variables under consideration and greatly aid in the task of understanding the interactions among the domain variables. Imagine for example a medical researcher, attempting to model a set of disease and symptom indicators in an effort to find ways of treating or preventing them from occurring in the future. Conditional independence statements can lend her significant insights to the interactions that may be present. Similarly, knowledge of conditional independencies can help a solid state physicist, attempting to model the interaction among tens of variables representing proportions of substances in a new manufacturing process, focus her attention to only the relevant ones during the different stages of the process.

More than simply aiding in focusing attention, as we already mentioned in section 2.7.3 there exist a number of algorithms today that can induce the causal structure of a domain given a set of conditional independence statements, under assumptions (Spirtes et al., 1993; Pearl, 2nd Ed., 1997), in the form of a Bayesian network. In this chapter we present two new approaches for inference of the structure of a BN from conditional independence tests. We first describe the GS algorithm, which is asymptotically faster than existing state-of-the-art approaches. It operates by identifying the local neighborhood of each variable in the Bayesian network as a preprocessing step, in order to facilitate the recovery of the exact structure around each variable in subsequent steps. This has the added advantage of being easier to verify and possibly correct *a posteriori* by an outside expert. For situations where the GS algorithm cannot be applied (*e.g.* in domains with large number of variables densely connected), we also provide Monte Carlo version of the algorithm which employs a constant number of randomized tests to ascertain the same result with high probability. The advantage of this is that it is able to operate in an "anytime" manner by allowing the user to adjust the

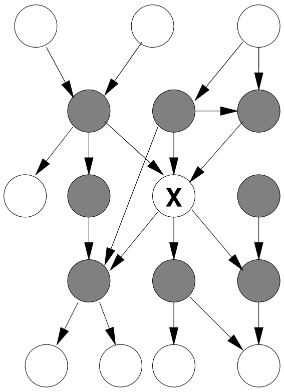


Figure 3.1: Example of a Markov blanket of variable X. The members of the blanket are shown shaded.

maximum number of tests to be done per structural decision. In addition, it is also able to operate in an online fashion by using Bayesian accumulation of evidence from multiple data sets, as they become available at runtime.

The assumptions we make are Causal Sufficiency, Causal Markov, and Faithfulness, defined in section 2.6. In addition, we assume no errors in the independence tests. In practice errors occur, and in order to make our proposed algorithms useful and practical we include heuristic steps that attempt to recover from simple errors by enforcing known consistency constraints such as acyclicity. In the latter respect, the algorithms here differ from ones such as the SGS, PC and IC.

3.1 The Grow-Shrink (GS) Markov Blanket Algorithm

The concept of the **Markov blanket** of a variable or a set of variables is central to the algorithms presented in this chapter. The idea itself is not new (for example, see Pearl (2nd Ed., 1997)). It is surprising, however, how little attention it has attracted in the context of Bayesian network structure learning for all its being a fundamental property of a Bayesian network. The definition of a Markov blanket is as follows: for any variable $X \in \mathcal{U}$, the Markov blanket $\mathbf{BL}(X) \subseteq \mathcal{U}$ is any set of variables such that for any $Y \in \mathcal{U} - \mathbf{BL}(X) - \{X\}$, $X \perp Y \mid \mathbf{BL}(X)$. In other words, $\mathbf{BL}(X)$ completely "shields" (d-separates) variable X from any other variable outside $\mathbf{BL}(X) \cup \{X\}$. The notion of a *minimal Markov blanket*, called a **Markov boundary**, is also introduced in Pearl (2nd Ed., 1997) and its uniqueness shown under certain conditions. The Markov boundary is not unique in certain situations, such as the equality of two variables. In our following discussion we will assume that the conditions necessary for its existence and uniqueness are satisfied and we will identify the Markov blanket with the Markov boundary, using the notation $\mathbf{B}(X)$ for the blanket of variable X from now on. It is illuminating to mention that, in the Bayesian network framework, the Markov blanket of a node X is easily identifiable from the graph: it consists of all parents, children and parents of children of

1. $\mathbf{S} \leftarrow \emptyset$.
2. While $\exists Y \in \mathcal{U} - \{X\}$ such that $Y \not\perp X \mid \mathbf{S}$, do $\mathbf{S} \leftarrow \mathbf{S} \cup \{Y\}$. **[Growing phase]**
3. While $\exists Y \in \mathbf{S}$ such that $Y \perp X \mid \mathbf{S} - \{Y\}$, do $\mathbf{S} \leftarrow \mathbf{S} - \{Y\}$. **[Shrinking phase]**
4. $\mathbf{B}(X) \leftarrow \mathbf{S}$.

Figure 3.2: The GS Markov Blanket Algorithm.

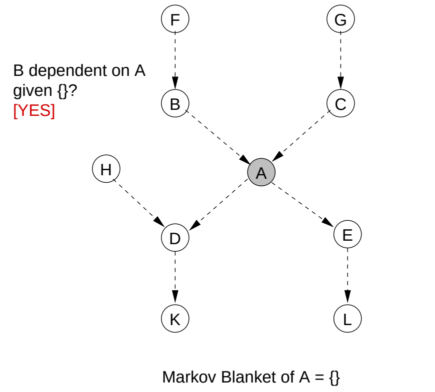
X. An example Markov blanket is shown in Fig. 3.1. Note that any node in the blanket, say Y, is dependent with X given $\mathbf{B}(X) - \{Y\}$ (assuming Faithfulness).

In Fig. 3.2 we present an algorithm, called GS (grow-shrink), for the recovery of the Markov blanket of X based on pairwise independence tests. It consists of two phases, a growing and a shrinking one, hence its name. Starting from an empty set $\mathbf{S}$, the growing phase adds variables to $\mathbf{S}$ as long as they are dependent with X given the current contents of $\mathbf{S}$. The idea behind this is simple: as long as the Markov blanket property of X is violated (*i.e.* there exists a variable in $\mathcal{U}$ that is dependent on X given the current $\mathbf{S}$), we add it to the current set $\mathbf{S}$ until there are no more such variables. In this process however, there may be some variables that were added to $\mathbf{S}$ that were really outside the blanket. Such variables are those that have been rendered independent from X at a later point when "intermediate" (d-separating) nodes of the underlying Bayesian net were added to $\mathbf{S}$. This observation motivates the shrinking phase, which identifies and removes these variables.

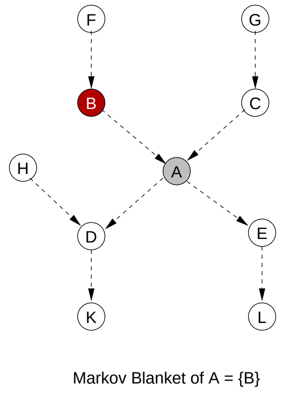
3.1.1 An illustrative example

We illustrate the operation of the GS Markov blanket algorithm step-by-step on an example BN. In Fig. 3.3(a) we show the underlying BN from which our data set is generated, with the edges of the DAG shown as dashed lines. As we mentioned above, we assume Faithfulness, so all independence tests that we conduct are assumed to return the correct result. In our example we illustrate the algorithm for the recovery of the Markov blanket of variable A. The algorithm does not specify an order for the variables, so for the sake of illustration we choose an arbitrary one, *e.g.* B, F, G, C, K, D, H, E, L. Here are the steps that the algorithm takes, with detailed explanations:

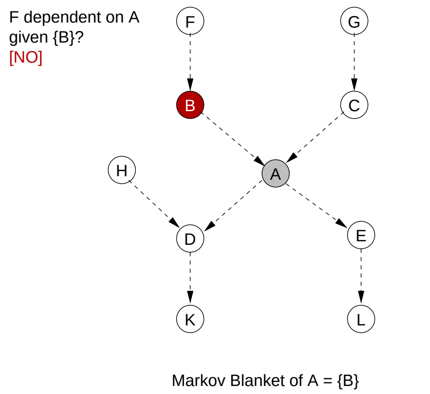
Growing phase Assuming the order that the variables are examined is B, F, G, C, K, D, H, E and L, the following events occur:



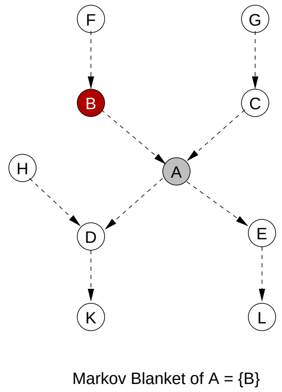
(a) Testing: $A \perp B \mid \{\}$.



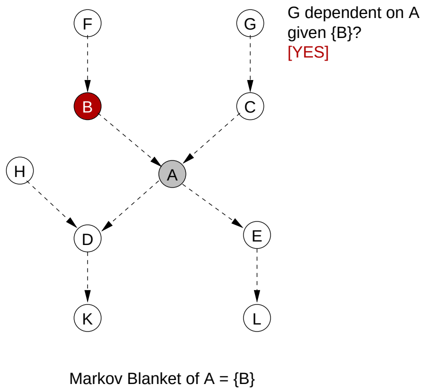
(b) Result: $A \not\perp B \mid \{\}$.



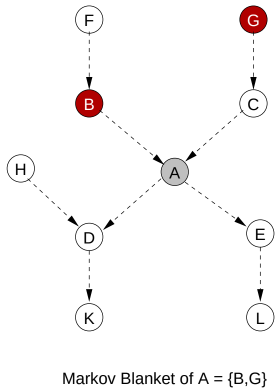
(c) Testing: $A \perp F \mid \{B\}$.



(d) Result: $A \perp F \mid \{B\}$.

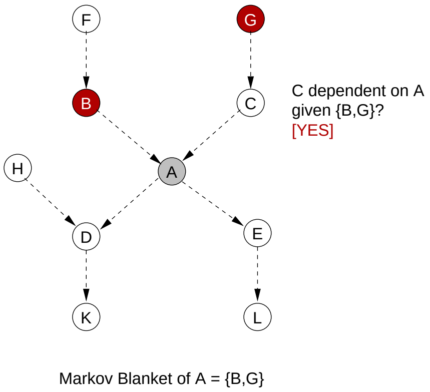


(e) Testing: $A \perp G \mid \{B\}$.

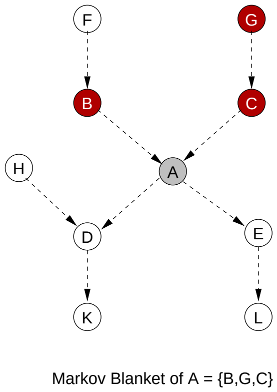


(f) Result: $A \not\perp G \mid \{B\}$.

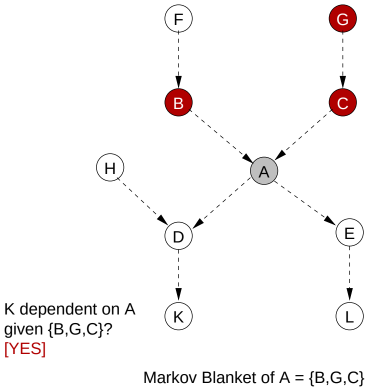
Figure 3.3: **Growing phase:** First two phases of the GS Markov blanket algorithm for node A, which attempt to add B, F and G to the running blanket of A. The structure of the original BN is shown with dashed lines.



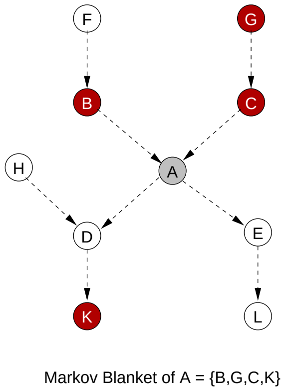
(a) Testing: $A \perp C \mid \{B,G\}$.



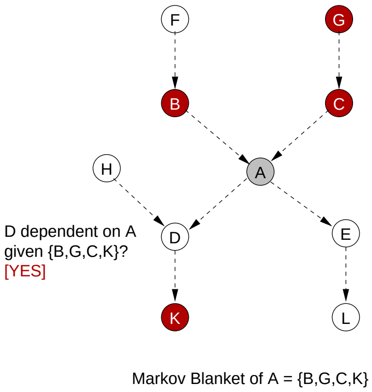
(b) Result: $A \not\perp C \mid \{B,G\}$.



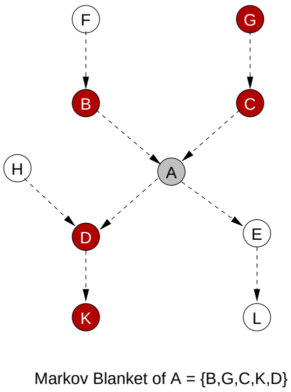
(c) Testing: $A \perp K \mid \{B,G,C\}$.



(d) Result: $A \not\perp K \mid \{B,G,C\}$.

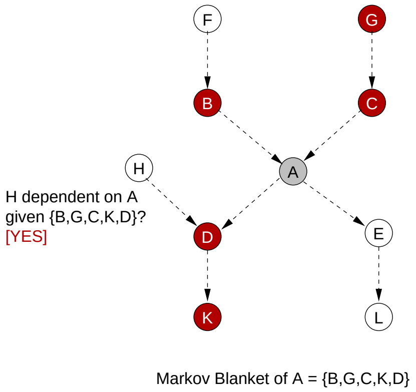


(e) Testing: $A \perp D \mid \{B,G,C,K\}$.

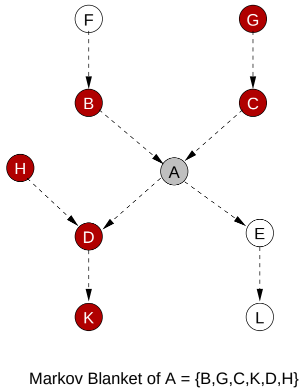


(f) Result: $A \not\perp D \mid \{B,G,C,K\}$.

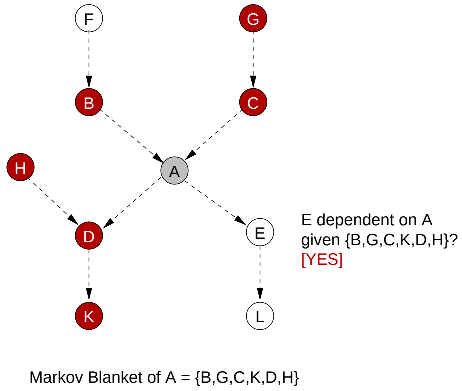
Figure 3.4: **Growing phase (continued):** Test C, K and D for inclusion to the current Markov blanket for A.



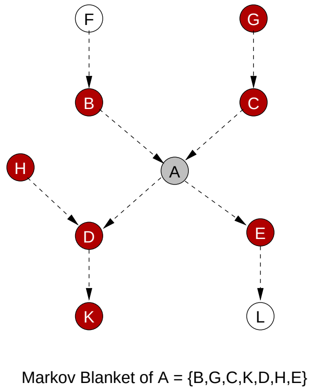
(a) Testing: $A \perp H \mid \{B,G,C,K,D\}$.



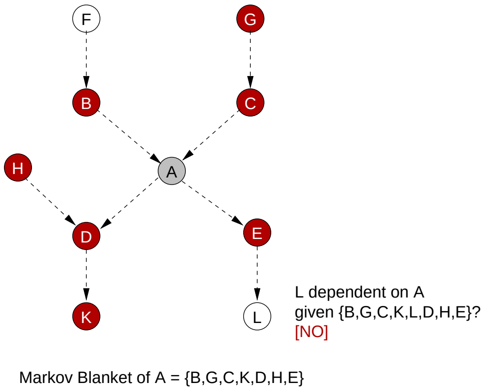
(b) Result: $A \not\perp H \mid \{B,G,C,K,D\}$.



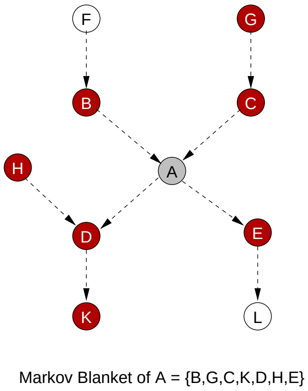
(c) Testing: $A \perp E \mid \{B,G,C,K,D,H\}$.



(d) Result: $A \not\perp E \mid \{B,G,C,K,D,H\}$.

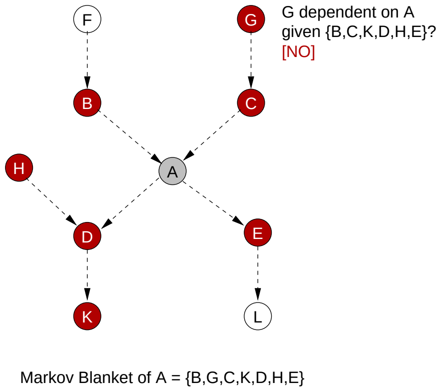


(e) Testing: $A \perp L \mid \{B,G,C,K,D,H,E\}$.

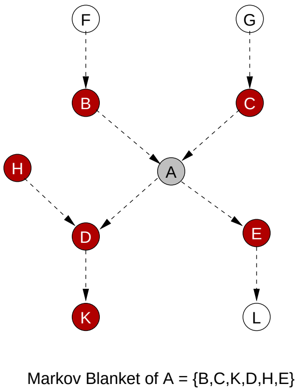


(f) Result: $A \perp L \mid \{B,G,C,K,D,H,E\}$.

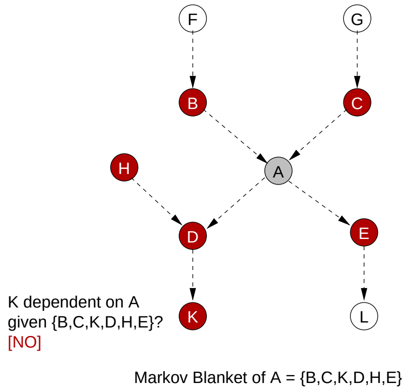
Figure 3.5: **Growing phase (continued):** Test H, E and L for inclusion to the current Markov blanket for A.



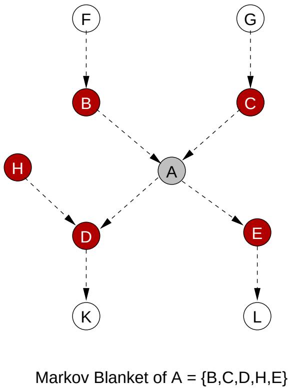
(a) Testing: $A \perp G \mid \{B, C, K, D, H, E\}$.



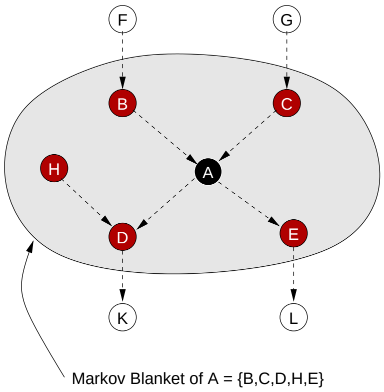
(b) Result: $A \perp G \mid \{B, C, K, D, H, E\}$.



(c) Testing: $A \perp K \mid \{B, C, K, D, H, E\}$.



(d) Result: $A \perp K \mid \{B, C, K, D, H, E\}$.



(e) Final Markov Blanket of A : $\{B, C, D, H, E\}$.

Figure 3.6: **Shrinking phase:** Test variables for removal from the current Markov blanket for A. Only G and K are removed.

1. Initially, the running Markov blanket of A is empty (denoted $\mathbf{S}$ in the pseudocode of Fig. 3.2). The first test of the growing phase is $A \perp B \mid \{\}$ which comes back false, so B is added to $\mathbf{S}$ which now becomes $\mathbf{S} = \{B\}$.
2. The next test is between A and F conditioned on the current value of $\mathbf{S}$, namely $A \perp F \mid \{B\}$. This comes back true, since F is blocked (d-separated) from A by B. Therefore F is not added to $\mathbf{S}$. $\mathbf{S} = \{B\}$.
3. Next G is tested for independence from A given $\mathbf{S} = \{B\}$, which comes back false. Therefore G is added to the running blanket even though it does not really belong to it—it was added simply because of our (unfortunate) variable ordering choice. (For example, had C been examined—and added to $\mathbf{S}$—before G, G would be blocked by it and excluded from $\mathbf{S}$.) $\mathbf{S} = \{B, G\}$.
4. Next the test $A \perp C \mid \{B, G\}$ returns false, causing C to be added to the blanket (correctly). $\mathbf{S} = \{B, G, C\}$.
5. In a similar fashion K and D are added to $\mathbf{S}$, in this order, making $\mathbf{S} = \{B, G, C, K, D\}$.
6. Next comes the turn of H. Because we assumed Faithfulness, the test $A \perp H \mid \{B, G, C, K, D\}$ returns false (in accordance with the rules of d-separation), so H is added to $\mathbf{S}$. $\mathbf{S} = \{B, G, C, K, D, H\}$.
7. Finally, at the end of the growing phase E is added to $\mathbf{S}$ while L is not, since it is blocked by E. $\mathbf{S} = \{B, G, C, K, D, H, E\}$.

Shrinking phase Assuming the same examining order, the following events occur:

1. $A \perp G \mid \{B, C, K, D, H, E\}$ returns true and G is removed from $\mathbf{S}$.
2. Similarly, $A \perp K \mid \{B, C, D, H, E\}$ is also true and K is also removed.
3. All other independence tests return false and thus no other variables are removed from $\mathbf{S}$. The shrinking phase terminates, and the correct Markov blanket of A, namely $\{B, C, D, H, E\}$ is returned.

Below we prove the correctness of the algorithm and give a timing analysis.

3.1.2 Proof of Correctness of the GS Markov Blanket Algorithm

By "correctness" we mean the ability of the algorithm to produce the Markov blanket of any variable in the original Bayesian network if all independence tests done during its course are assumed to be correct and the underlying probability distribution is Faithful to the original BN structure.

We first note that there does not exist any variable $Y \in \mathbf{B}(X)$ at the end of the growing phase that is not in $\mathbf{S}$. The proof is by contradiction: take $Y \in \mathcal{U}$ such that $Y \in \mathbf{B}(X)$ but $Y \notin \mathbf{S}$. Then either Y is a direct neighbor of X (direct ancestor or direct descendant), or it is a direct ancestor of a direct descendant of X. In the former case $Y \not\perp X \mid \mathbf{T}$ for any $\mathbf{T} \subseteq \mathcal{U} - \{X,Y\}$. Therefore necessarily $Y \in \mathbf{S}$ at the end of the growing phase. In the latter case, either the common child of X and Y, say Z, is in $\mathbf{S}$ or is not. If $Z \in \mathbf{S}$ then Y must also be in $\mathbf{S}$, since $X \not\perp Y \mid \mathbf{S}$. If $Z \notin \mathbf{S}$ then we have a contradiction by the same argument as above since Z is a direct descendant of X.

For the correctness of the shrinking phase, we have to prove two things: that we never remove any variable Y from $\mathbf{S}$ if $Y \in \mathbf{B}(X)$, and that all variables $W \notin \mathbf{B}(X)$ are removed.

For the former, suppose Y is the first variable in $\mathbf{B}(X)$ that we are attempting to remove. Then Y is either a direct neighbor of X or the direct ancestor of a direct descendant of X, say Z. In the former case, since $Y \not\perp X \mid \mathbf{T}$ for any $\mathbf{T} \subseteq \mathcal{U}$, Y cannot be removed, leading to a contradiction. In the latter case, since Y is the first variable in $\mathbf{B}(X)$ to be removed, then Z must be in $\mathbf{S}$ and therefore, since $Y \not\perp X \mid \mathbf{S}$, Y cannot be removed.

Finally we need to show that there is no variable W in $\mathbf{S}$ at the end of the shrinking phase such that $W \notin \mathbf{B}(X)$. Suppose the opposite. Then since $\mathbf{B}(X) \subseteq \mathbf{S}$ as shown above, then $W \perp X \mid \mathbf{S}$, and W will necessarily be removed by the algorithm during this phase.

3.1.3 Complexity Analysis of the Markov Blanket Algorithm

Each dependence test takes $O(n|\mathcal{D}|)$ time, where $\mathcal{D}$ is the set of examples input to the algorithm. This is consumed in constructing the table of counts, for each value combination of the variables included in the test that exist in the data set. We make the assumption that combinations of variable values that do not appear in the data are improbable and their probability is approximated by 0 in the absence of other information. This assumption makes the test linear in the number of examples (and not exponential in the number of variables as would be the case if examples existed for all possible value combinations of $\mathcal{U}$). Each dependence test uses $O(|\mathcal{D}|)$ space at worst to store the counters for each variable value combination of the conditioning set that appears in the data.

Frequently the number of independence tests conducted is reported as a measure of the performance of Bayesian net reconstruction algorithms, *e.g.* Spirtes et al. (1993); Cheng et al. (1997). To determine the number of tests in this algorithm, we assume that the loops in steps 2 and 3 go through eligible variables in an unspecified but fixed order. In step 2, one complete pass through all variables will add at least the parents and children of X. A second pass will add all parents of all children of X, thus including all members of $\mathbf{B}(X)$. A possible third pass will not add any other members and step 2 will terminate. Thus, step 2 conducts $O(n)$ tests at worst. In step 3, a single pass through the variables in $\mathbf{S}$ will remove all members not in $\mathbf{B}(X)$ since $\mathbf{S}$ already contains $\mathbf{B}(X)$.

Therefore the entire algorithm is $O(n)$ in the number of independence tests.

3.1.4 Discussion

Although the algorithm is already efficient ($O(n)$ in the number of tests), it may benefit from certain optimizations. One may attempt to use a heuristic while adding the members of $\mathbf{B}(X)$ in the first pass of step 2, thus eliminating one of the two remaining passes. Using mutual information in the choice of the next variable to examine in step 2, for example, may help improve the performance of the algorithm in practice. The idea is to try to add to $\mathbf{S}$ first these variables for which the mutual information with X is the highest. Unfortunately no guarantees can be made of completely eliminating a pass this way because of certain cases where the highest mutual information variables are not members of the blanket. These cases are not very common and require the construction of elaborate distributions to demonstrate their existence, but theoretically may occur nevertheless.

An opportunity for computational amortization exists, stemming from the (easily proved) fact that $Y \in \mathbf{B}(X) \Leftrightarrow X \in \mathbf{B}(Y)$. Using this property, if we are interested in more than one variable's blanket and we obtain them in a sequential manner (as in the GS algorithm described below), we can initialize a variable's blanket, say X, with those variables Y for which we have computed the blanket in previous steps, by checking that they already include X in their blankets.

3.2 Grow-Shrink (GS) Algorithm for Bayesian Network Induction

The recovery of the local structure around each node is greatly facilitated by the knowledge of the nodes' Markov blankets. What would normally be a daunting task of employing dependence tests conditioned on an exponential number of subsets of large sets of variables—even though most of their members may be irrelevant—can now be focused on the Markov blankets of the nodes involved, making structure discovery faster. We present below the plain version of the GS algorithm that utilizes blanket information for inducing the structure of a Bayesian network. At a later point of this chapter, we will present a robust, randomized version that has the potential of being faster, as well as being able to operate in an "anytime" manner.

The GS algorithm is shown in Fig. 3.7. $\mathbf{N}(X)$ represents the direct neighbors of X. In the algorithm description, step 2 determines which of the members of the blanket of each node are actually direct neighbors (parents and children). The way it does that is by a series of dependence tests between X and Y conditioned on all subsets of the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$. Assuming, without loss of generality, that $\mathbf{B}(X) - \{Y\}$ is the smaller set, if any of these tests are successful in separating (rendering independent) X from Y, the algorithm determines that there is no direct connection between them. That would happen when the conditioning set $\mathbf{S}$ includes all parents of X and no common children of X and Y. It is interesting to note the two motivations behind selecting the smaller set to condition on: the first, of course, is speed, since conditioning on a set $\mathbf{S}$ might take $O(2^{|\mathbf{S}|})$ time. However, the second reason stems from reliability: a

1. [Compute Markov Blankets]
For all $X \in \mathcal{U}$, compute the Markov blanket $\mathbf{B}(X)$.

2. [Compute Graph Structure]
For all $X \in \mathcal{U}$ and $Y \in \mathbf{B}(X)$, determine Y to be a direct neighbor of X if X and Y are dependent given $\mathbf{S}$ for all $\mathbf{S} \subseteq \mathbf{T}$, where $\mathbf{T}$ is the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$.

3. [Orient Edges]
For all $X \in \mathcal{U}$ and $Y \in \mathbf{N}(X)$, orient $Y \to X$ if there exists a variable $Z \in \mathbf{N}(X) - \mathbf{N}(Y) - \{Y\}$ such that Y and Z are dependent given $\mathbf{S} \cup \{X\}$ for all $\mathbf{S} \subseteq \mathbf{T}$, where $\mathbf{T}$ is the smaller of $\mathbf{B}(Y) - \{X, Z\}$ and $\mathbf{B}(Z) - \{X, Y\}$.

4. [Remove Cycles]
Do the following while there exist cycles in the graph:

- Compute the set of edges $\mathbf{C} = \{X \to Y$ such that $X \to Y$ is part of a cycle$\}$.
- Remove from the current graph the edge in $\mathbf{C}$ that is part of the greatest number of cycles, and put it in $\mathbf{R}$.

5. [Reverse Edges]
Insert each edge from $\mathbf{R}$ in the graph in reverse order of removal in Step 4, reversed.

6. [Propagate Directions]
For all $X \in \mathcal{U}$ and $Y \in \mathbf{N}(X)$ such that neither $Y \to X$ nor $X \to Y$, execute the following rule until it no longer applies: If there exists a directed path from X to Y, orient $X \to Y$.

Figure 3.7: The GS algorithm.

conditioning set $\mathbf{S}$ causes the data set to be split into $2^{|\mathbf{S}|}$ partitions, so smaller conditioning sets cause the data set to be split into larger partitions and make dependence tests more reliable.

We give an illustration of the operation of step 2 on an example BN that we encountered before in Fig. 2.2, reproduced in Fig. 3.8 for convenience. To determine whether there is an edge between C and D for example, the algorithm uses their blankets $\mathbf{B}(C) = \{A, B, D, E, F\}$ and $\mathbf{B}(D) = \{B, C, F\}$. Since $|\mathbf{B}(D)| < |\mathbf{B}(C)|$, step 2 conditions on all subsets of $\mathbf{B}(D) - \{C\} = \{B, C, F\} - \{C\} = \{B, F\}$:

1. $C \not\perp D \mid \{\}$ since there is a path of dependence $C - B - D$.
2. $C \perp D \mid \{B\}$ since there path $C - B - D$ is blocked by conditioning on B.
3. $C \not\perp D \mid \{F\}$ since there is a path of dependence $C - B - D$.
4. $C \not\perp D \mid \{B, F\}$ since there are two paths of dependence, namely $C - B - D$ and $C - F - D$.

Because $C \perp D \mid \{B\}$ (item 2 above), the algorithm correctly concludes that there is no direct edge $C - D$.

Step 3 of the algorithm exploits the fact that two variables that have a common descendant become dependent when conditioning on a set that includes any such descendant. Since the direct neighbors of X and Y are

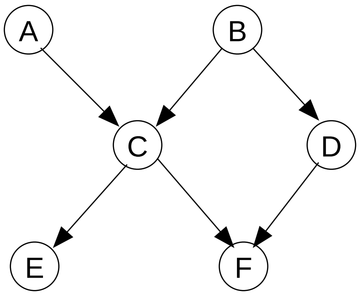


Figure 3.8: Example BN used to illustrate the operation of the GS algorithm.

known from step 2, we can determine whether a direct neighbor Y is a parent of X if there exists another node Z (which would also be a parent) such that all attempts to separate Y and Z by conditioning on a subset of the blanket of Y that includes X, fail (assuming here that $\mathbf{B}(Y)$ is smaller than $\mathbf{B}(Z)$). If the directionality is indeed $Y \to X \leftarrow Z$, there should be no such subset since, by conditioning on X, there exists a permanent dependency path between Y and Z. This would not be the case if Y were a child of X.

Steps 1–3 essentially correspond to steps 1–2 of the SGS algorithm (see Fig. 2.5), adapted to take advantage of knowledge of the neighborhoods of the nodes, produced in step 1 of the GS algorithm.

It is possible that an edge direction will be wrongly determined during step 3 due to non-representative or noisy data. This may lead to directed cycles in the resulting graph, which are illegal. It is therefore necessary to remove those cycles by identifying the minimum set of edges than need to be reversed for all cycles to disappear. This problem is closely related to the *Minimum Feedback Arc Set* problem (*MFAS*), which consists of identifying a minimum set of edges that need to be removed from a graph that possibly contains directed cycles, in order for all such cycles to disappear. Here instead of removing those edges we want to reverse them. Unfortunately, the *MFAS* problem is NP-complete in its generality (Jünger, 1985) and the weighted edges instance as it applies here is also NP-complete (see theorem in Appendix B). In the algorithm above we introduce a heuristic for its solution that is based on the number of cycles that an edge that is part of a cycle is involved in.

Not all edge directions can be determined during the last two steps. For example, nodes with a single parent or multi-parent nodes (called *colliders*) whose parents are directly connected (called *shielded* colliders) do not apply to step 3, and steps 4 and 5 are only concerned with already directed edges. As an example, was there an edge $A \to B$ in the BN of Fig. 3.8, step 3 would not apply and the direction of the edge between A and B (the existence of which would had been established in step 2) would not be determined by it. Step 6 attempts to rectify that in a way similar to that in the SGS (Fig. 2.5). This is done through orienting edges in a way that does not introduce a cycle, if the reverse direction necessarily does. It is not obvious that if the direction $X \to Y$ produces a cycle in an otherwise acyclic graph, the opposite direction $Y \to X$ will not also be involved in some other cycle. However, this is the case. The proof of this is simple is presented below in the proof of correctness of the GS algorithm.

3.2.1 Proof of Correctness of the GS Algorithm

By "correctness" we mean that the Bayesian network resulting from the above procedure is equivalent to the original one representing the physical process by which the data was created. To do that, it suffices to show two things:

1. All nodes in the resulting network have the same neighbors as the original Bayesian network, and
2. All unshielded colliders in the resulting network have the same parents as the original network.

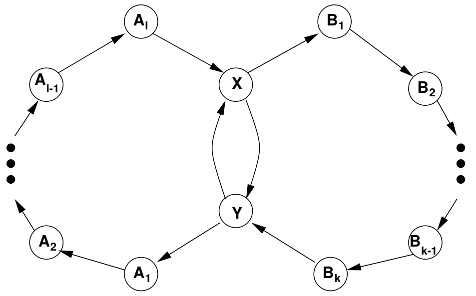
These requirements stem from a theorem by Verma and Pearl (1990) that states that two Bayesian networks are equivalent if and only if each node has the same neighbors and the two networks have the same "V-structures," *i.e.* colliders with the same parents (and, by requirement 1, the same children as well).

To prove correctness we assume Causal Sufficiency, Causal Markov and Faithfulness (Section 2.6).

In the following proof of correctness of step 2, we assume that $|\mathbf{B}(X)| \leq |\mathbf{B}(Y)|$, without loss of generality. Thus conditional tests between X and Y will involve conditioning on all subsets of $\mathbf{B}(X) - \{Y\}$. To show that the network resulting from the above algorithm satisfies the first requirement, we observe that any two nodes X and Y in the original network that are directly connected remain dependent upon conditioning on any set $\mathbf{S} \subseteq \mathbf{B}(X) - \{Y\}$. Therefore all tests between two such variables in step 2 will be positive and the algorithm will correctly include Y in $\mathbf{N}(X)$. In the case that X is not directly connected to Y, that means that Y is the parent of a number of children of X, say set $\mathbf{T}$, since it is a member of $\mathbf{B}(X)$. Therefore conditioning on a set that includes all parents of X and excludes $\mathbf{T}$ will block all paths from X to Y.

The proof of the second requirement is similar. The only difference is that instead of direct connectedness of two direct neighbors of X, say Y and Z, we are trying to ascertain connectedness through conditioning on any subset of $\mathbf{B}(Y)$ that necessarily includes X (without loss of generality we assume that $|\mathbf{B}(Y)| \leq |\mathbf{B}(Z)|$). If Y and Z are both parents of X, they would be dependent upon conditioning on any such set via the open path through X. If Y is a child of X, then conditioning on X blocks one of the paths to Z, and any other paths to Z are blocked by conditioning on the remaining parents of Y.

Since we assumed Faithfulness, the tests in step 3 will correct, and steps 4 and 5 will not attempt to eliminate any cycles. However, in the presence of errors in the independence tests (*i.e.* failure of Faithfulness), they will, and we need to prove that they do not produce an illegal (*i.e.* cyclical) structure. This is done as follows. The algorithm removes a number of edges until there are no remaining cycles. These edges can then be added in reverse direction in step 5 without introducing cycles. The correctness of this relies on the same observation that that the last step does. We need to show that the incremental addition of edges for which the reverse direction necessarily creates a cycle in the graph, does not itself introduce any cycles. The proof is by contradiction. The original graph, as the result of step 4, is acyclic. Assume that both $X \to Y$ and $Y \to X$ will produce a cycle when added to the set of edges (each individually) in an otherwise acyclic graph *e.g.* assume that adding $X \to Y$ will create cycle $X \to Y \to A_1 \to A_2 \to \cdots \to A_l \to X$ and adding $Y \to X$ will create cycle $Y \to X \to B_1 \to B_2 \to \cdots \to B_k \to Y$, as shown in the figure below.



However, that arrangement necessarily implies that the original graph already contained a cycle, namely $X \to B_1 \to \cdots \to B_k \to Y \to A_1 \to \cdots \to A_l \to X$, which is a contradiction. Therefore exactly one of $X \to Y$ and $Y \to X$ can be added to the graph without introducing a cycle and the next iteration still satisfies the inductive assumption that the graph is acyclic.

As mentioned above, the same argument may be used to show the correctness of the last step.

3.2.2 Complexity Analysis of the GS Algorithm

Since the Markov blanket algorithm involves $O(n)$ conditional independence (CI) tests, step 1 involves $O(n^2)$ tests. If $b = \max_X(|\mathbf{B}(X)|)$, step 2 does $O(nb2^b)$ CI tests. At worst $b = O(n)$, implying that the set of examples $\mathcal{D}$ was produced by a dense original Bayesian network. If we know that the upward branching factor is less than or equal to u and the downward branching factor less than equal to d, then $b \leq u + d + du$, which is a constant. Step 3 does $O(nb^2 2^b)$ CI tests, and the exponential factor may be limited similarly in the presence of branching factor information. Steps 4 and 5 do not do any independence tests. Checking for the existence of cycles in step 4 takes $O(m(n+m))$ time by employing a standard depth-first traversal of the graph for each existing edge (m is the number of edges in the network). Step 5 is $O(m)$ in time. The last step can be implemented in a straightforward manner in $O(nb(n+m))$ time using depth-first traversal, which does not affect the asymptotic time of the algorithm.

The total number of CI tests for the entire algorithm is therefore $O(n^2 + nb^2 2^b)$ or $O(m^2 + nmb + (n^3 + n^2 b^2 2^b)\,|\mathcal{D}|)$ time. In the worst case, *i.e.* when $b = O(n)$ and $m = O(n^2)$, this becomes $O(n^4 2^n\,|\mathcal{D}|)$ time. Under the assumption that b is bounded by a constant, this algorithm is $O(n^2)$ in the number of CI tests or $O(m^2 + n^3\,|\mathcal{D}|)$ time.

3.2.3 Discussion

The main advantage of the algorithm comes through the use of Markov blankets to restrict the size of the conditioning sets. The Markov blankets may be incorrect. The most likely potential error is to include too many nodes for reasons that are explained below. This emphasizes the importance of the "direct neighbors" step (step 2) which removes nodes that were incorrectly added during the Markov blanket computation step due to conditioning on large set of variables. The problem is the following: conditioning on n binary variables requires 2^n dependence tests. Each such test compares two histograms, the histogram representing the probability distribution of binary variable X given the conditioning set $\mathbf{S}$ and the histogram representing the distribution of X given $\mathbf{S} \cup \{Y\}$. The two variables X and Y are pronounced dependent if for any of the $2^{|\mathbf{S}|+1}$ configurations of the set $\mathbf{S} \cup \{Y\}$, the corresponding histograms are "different enough." However, given a limited number of samples, if we are using the same set of samples for all tests, the probability that one of these tests will be positive and Y will be added to the blanket of X even if the two variables are not really dependent is increasing rapidly with increasing conditioning set size $|\mathbf{S} \cup \{Y\}|$. The "direct neighbors" step, which does a number of dependence tests between X and Y and declares them direct neighbors only if *all* these tests have high confidence, helps identify and correct these potential errors in the preceding Markov blanket phase.

3.3 Randomized Version of the GS Algorithm

The GS algorithm presented above is appropriate for situations where the maximum size of the Markov blanket of each variable under consideration is sufficiently small, since it depends on it through an exponential factor. While it is reasonable to assume that in many real-life problems this may be the case, certain ones, such as Bayesian image retrieval in computer vision, may employ finer representations. In these cases the variables used may depend in a direct manner on many others. For example, one may choose to use variables to characterize local texture in different parts of an image. If the resolution of the mapping from textures to variables is fine, direct dependencies among those variables may be plentiful, suggesting a dense underlying network. In these cases, it may be prohibitively expensive to employ the exponential number of tests such as those done in the plain GS algorithm.

Another problem of the GS algorithm presented above, and one that has plagued independence-test based algorithms for Bayesian network structure induction in general, is that their decisions are based on a single or a few tests, making them prone to errors due to possible noise in the data. It would therefore be advantageous to allow multiple tests to influence a decision before determining the existence of an edge or its direction in the resulting network.

The following version of the GS algorithm addresses these two problems. First, by executing a fixed number of tests during the determination of the existence of an edge, using conditioning sets randomly drawn from the smallest of the two blankets, the time allocated to each edge may be finely controlled so that the algorithm is able to execute even for problems for which blanket sizes are large. An additional advantage of the approach is that it is now able to operate in an "anytime" manner, which is useful in real-time applications such as robot vision (another anytime algorithm can be found in Spirtes (2001)). Second, using Bayesian evidence accumulation, each structural decision depends only partly on the outcome of each single test (although the tests have to be appropriately weighted—see the discussion below), and the algorithm can operate in an "online" fashion, processing each new data set as soon as it is available. This kind of applications, for example robot vision where new images are available every few microseconds, is indeed one of the main targets of the algorithm.

This version of the algorithm is presented in Fig. 3.9. The algorithm computes the posterior probabilities that certain variables X and Y are direct neighbors and whether a possible link between them is directed as $Y \to X$ or $X \to Y$ after seeing a set of M data sets (denoted here as a vector of data sets) Ξ_M, that are assumed to be independently drawn. Each data set $\xi_m, i = 1, \ldots, M$ contains a number of examples.

The derivations for the formula used above to compute the posterior probabilities is presented in Appendix A. The same formula is used in two places in the algorithm, in updating the posterior probability of the existence of an edge and also for the corresponding probability on the direction of those edges that were determined to be present. The use of the parameter G, which acts to weigh each test, marks its difference from straightforward posterior application of formulas such as those found in Pearl (2nd Ed., 1997). G roughly characterizes the connectivity of a node in terms of the size of its blanket, ranging from 0 (blanket size 1) to 1 (blanket size very large, tending to infinity). We can intuitively see its influence on the posterior probability by examining limiting cases for the parameters occurring in the formula, which is repeated below for the reader's convenience:

$$p \leftarrow \frac{pd}{pd + (1-p)(G+1-d)}.$$

Note that since the true size of the Markov blanket is not known this posterior probability is an approximation (given our assumptions) and not a bound on the true posterior probability.

We will examine the application of the formula to the existence of a direct link between two nodes X and Y. The application of the formula to the directionality of a link is similar. For simplicity we will assume that X has the smaller blanket of the two nodes, and therefore G is computed using $|\mathbf{B}(X)|$ *i.e.* $\mathbf{T} = \mathbf{B}(X) - \{Y\}$ in Fig. 3.9. We first look at the case when the independence test $d = \Pr(X \not\perp Y \mid \mathbf{S}, \xi_m) = 0$. In this case, the posterior probability p becomes 0, as expected, since the hypothesis of a direct link between X and Y cannot possibly support this evidence. In the case that $d = 1$, the above formula becomes

$$p \leftarrow \frac{p}{p + (1-p)G}.$$

We look at two limiting cases for G. If $G \approx 0$ (*i.e.* $\mathbf{B}(X) = \{Y\}$), p becomes ≈ 1. This is reasonable

1. [Compute Markov Blankets] (same as plain GS)
For all $X \in \mathcal{U}$, compute the Markov blanket $\mathbf{B}(X)$.

2. [Compute Graph Structure]
For each $X \in \mathcal{U}$ and $Y \in \mathbf{B}(X)$ do:

- Set $p \leftarrow \frac{1}{2}$.
- Set $\mathbf{T}$ to be the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$.
- Let $G \leftarrow G(X,Y) = 1 - (\frac{1}{2})^{|\mathbf{T}|}$.
- For each data set ξ_m, $i = 1, \ldots, M$, execute the following:
  - Set $\mathbf{S}$ to be a randomly chosen subset of $\mathbf{T}$.
  - Compute $d = \Pr(X \not\perp Y \mid \mathbf{S}, \xi_m)$.
  - Update the posterior probability p using the recursive formula
$$p \leftarrow \frac{pd}{pd + (1-p)(G+1-d)}$$
- Set $\Pr(Y \in \mathbf{N}(X)) = \Pr(X \in \mathbf{N}(Y)) = p$.
- Assign Y to be a member of $\mathbf{N}(X)$ and X to be in $\mathbf{N}(Y)$ if and only if $p > \frac{1}{2}$.

3. [Orient Edges]
For each $X \in \mathcal{U}$, $Y \in \mathbf{N}(X)$ do:

- Set $Q \leftarrow \frac{1}{2}$.
- Do for each $Z \in \mathbf{N}(X) - \mathbf{N}(Y) - \{Y\}$:
  - Set $q \leftarrow \frac{1}{2}$.
  - Set $\mathcal{U}$ to be the smaller of $\mathbf{B}(Y) - \{X, Z\}$ and $\mathbf{B}(Z) - \{X, Y\}$.
  - Let $G \leftarrow G(Y,Z) = 1 - (\frac{1}{2})^{|\mathcal{U}|}$.
  - For each data set ξ_m, $i = 1, \ldots, M$, execute the following loop:
    * Set $\mathbf{S}$ to be a randomly chosen subset of $\mathcal{U}$.
    * Compute $d = \Pr(Y \not\perp Z \mid \mathbf{S} \cup \{X\}, \xi_m)$.
    * Update the posterior probability q using the recursive formula
$$q \leftarrow \frac{qd}{qd + (1-q)(G+1-d)}$$
  - Update $Q \leftarrow \frac{Q(1-q)}{Q(1-q)+(1-Q)(1-G+q)}$.
- Set $\Pr(Y \to X) = 1 - Q$.

For each $X \in \mathcal{U}$, $Y \in \mathbf{N}(X)$ do:

- Assign direction $Y \to X$ if $\Pr(Y \to X) > \Pr(X \to Y)$.
- Assign direction $X \to Y$ if $\Pr(Y \to X) < \Pr(X \to Y)$.

4. [Remove Cycles]
Do the following while there exist cycles in the graph:

- Compute the set of edges $\mathbf{C} = \{X \to Y$ such that $X \to Y$ is part of a cycle$\}$.
- Remove the edge $X \to Y$ in $\mathbf{C}$ that such that $\Pr(X \in \mathbf{N}(Y)) \Pr(X \to Y)$ is minimum and put it in $\mathbf{R}$.

5. [Reverse Edges] (same as plain GS)
Insert each edge from $\mathbf{R}$ in the graph, reversed.

6. [Propagate Directions] (same as plain GS)
For all $X \in \mathcal{U}$ and $Y \in \mathbf{N}(X)$ such that neither $Y \to X$ nor $X \to Y$, execute the following rule until it no longer applies: If there exists a directed path from X to Y, orient $X \to Y$.

Figure 3.9: The randomized GS algorithm

since the high confidence of the dependence between X and Y combined with the absence of any other node in the blanket of X constitutes a direct indication of an edge between the two nodes. In case $G = 1$ (*i.e.* $|\mathbf{B}(X)| \to \infty$), a test indicating the dependence of X and Y conditioned on a random subset of $\mathbf{B}(X)$ is very weak evidence of a link between them, since the likelihood that a common ancestor will be absent or a common descendant will be present in the conditioning set (note that X is in $\mathbf{B}(Y)$ and vice versa) is high, and that would explain the evidence (the dependence) without the necessity of a link between the two nodes. Accordingly, the value of p does not change in this case.

The formulas in step 3 (Orient Edges) are seemingly complicated but they follow the same rationale. The quantity q that is computed in the loop is the probability that Y and Z, both direct neighbors of X, are dependent conditioned on a subset that contains X. However, the determination of whether $Y \to X$ or $Y \leftarrow X$ is an OR over all other direct neighbors Z. Our iterative formula that is based on multiple tests computes the probability that *all* tests are true *i.e.* an AND of tests. Therefore we use DeMorgan's law to convert the OR of tests into an AND of the negations of these tests. From this observation it is easy to see that Q computes the posterior probability that $Y \not\to X$.

Finally, as a general note, we observe that the robustness of this version of the algorithm is at least as great as the one of the plain GS algorithm, given the same number of tests. The additional robustness comes from the use of multiple weighted tests, leaving for the end the "hard" decisions that involve a threshold (*i.e.* comparing the posterior probability with a threshold, which in our case is $1/2$.

3.4 Experimental Results

In this section we present results using both synthetic and real-world datasets. The experiments demonstrate two main points. First, that both the plain and randomized GS algorithms perform better than hill-climbing approaches and the PC algorithm along certain dimensions of performance. And second, that the randomized version can execute much faster in cases of large neighborhood sized networks while maintaining good error rates compared to the plain GS algorithm.

Regarding the randomized version of the GS algorithm experiments, the assumption we use in the derivation of the formulas is that a new, independently drawn set of examples ξ_m is used with each dependence test conducted. However, given that data is frequently expensive and/or scarce, in our experiments we use the same data in all tests running the risk of overfitting the dataset, in order to demonstrate the performance of the algorithm under these adverse, but more realistic conditions. In the case that data is plentiful (such as in a robot vision application where capturing a new image is cheap), the procedure would split the dataset into subsets of examples randomly, and use one subset for each dependence test it makes.

In the GS algorithms presented in this chapter we employ standard chi-square (χ^2) conditional dependence tests in order to compare the histograms $\widehat{P}(X)$ and $\widehat{P}(X \mid Y)$. The χ^2 test gives us the probability of error of assuming that the two variables are dependent when in fact they are not (type II error of a dependence test), *i.e.* the probability that the dataset values for X and Y would be obtained by chance given that X and Y are independent. If this probability is very small (*e.g.* smaller than 0.10 or 0.05) then we conclude that X and Y are dependent. Equivalently, we can compare one minus this probability with 0.95; this is an implicit threshold τ that is involved in each dependence test, indicating how certain we wish to be about the correctness of the test without unduly rejecting dependent pairs, something always possible in reality due to the presence of noise. In our experiments we used both $\tau = 0.90$ and $\tau = 0.95$.

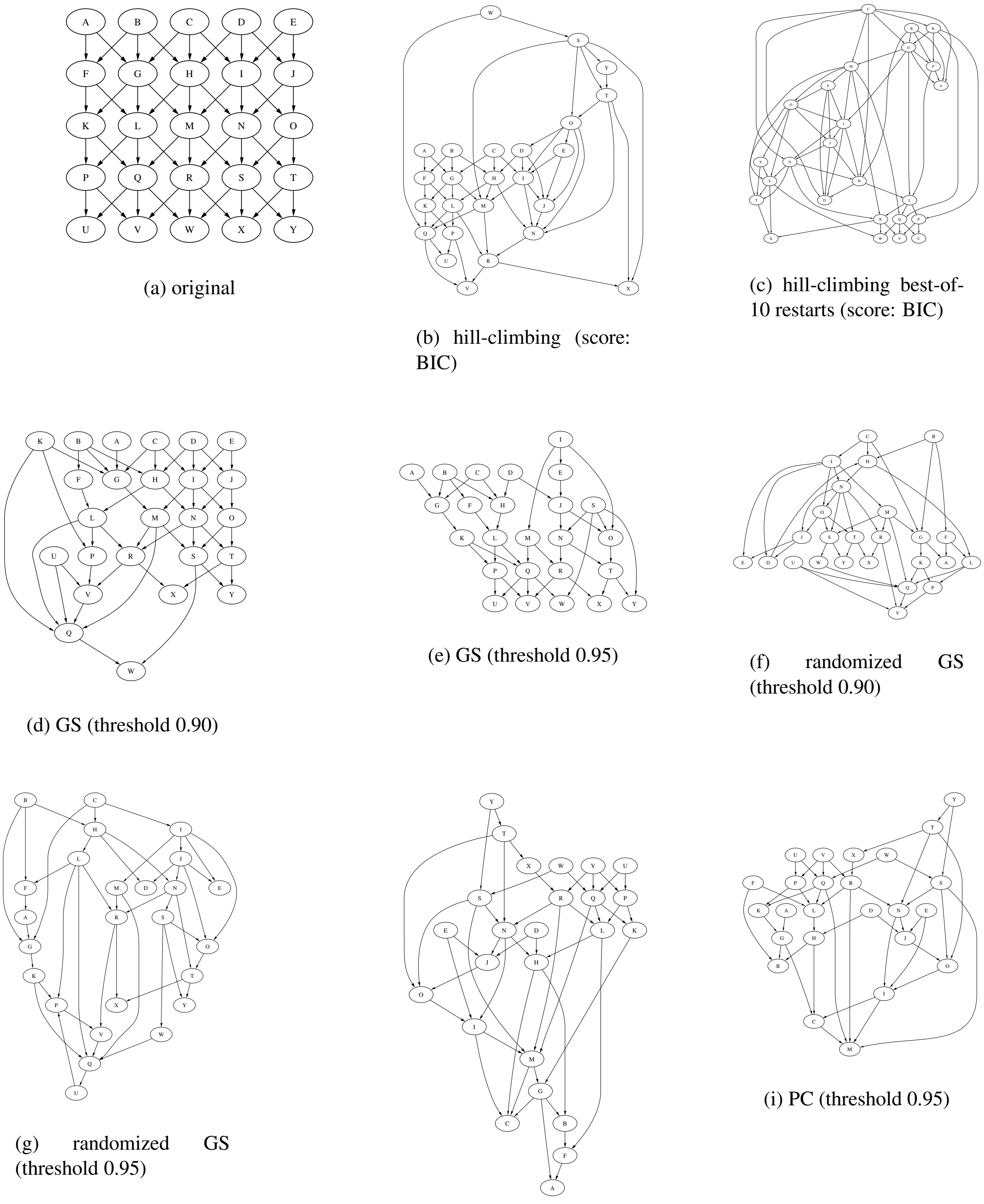
One important note we have to make here regarding the following experiments is that in certain cases it may not be possible to determine the directionality of all edges *e.g.* in the edge set $\{A \to C, B \to C, A \to B\}$. Since for some of the following experiments we need to calculate the data likelihood of the dataset with respect to a BN structure, we had to somehow decide on the direction of all edges. Our solution to this problem is to randomly decide on the direction of these edges in the cases where their direction cannot be determined by any independence test. Note that this is done for both the two versions of the GS and the PC algorithm. The inability of uniquely determining the direction of certain edges is not an uncommon problem and is the result of statistical indistinguishability of the two possible directions of an edge. In this sense, the GS and PC algorithms are "handicapped" by necessity in that way in the following experiments. The hill-climbing algorithm does not have this problem, since at any given point in time during the hill-climbing process the directions of all edges are determined.

3.4.1 Synthetic Datasets

Reconstruction of an example (artificially constructed) network using different techniques is shown in Fig. 3.10.

We test the effectiveness of the algorithms through the following procedure: we generate a random rectangular network of specified dimensions and up/down branching factor, where the nodes are arranged in a regular grid of those dimensions. The up branching factor specifies the number of parents of each node directly above it, excluding nodes near the left and right border of the grid, and on the top row. An example of a BN of dimensions 5×5 and up and down-branching factors of 3 is shown in Fig. 3.10(a). The down branching factor is specified similarly as the number of children of all nodes excluding border nodes and the bottom row. A number of examples are drawn from that network using logic sampling (Henrion, 1988) and are used as input to the algorithm under test. The resulting networks can be compared with the original ones along dimensions of data likelihood of the training set, number of edges returned that also appear in the original BN, number of edges returned that do not appear in the original BN, the number of edges correctly directed (out of those correctly identified).

Below we comment on each of the networks that result from the largest data set used (20,000 data points) by the algorithms used in this evaluation.



(a) original

(b) hill-climbing (score: BIC)

(c) hill-climbing best-of-10 restarts (score: BIC)

(d) GS (threshold 0.90)

(e) GS (threshold 0.95)

(f) randomized GS (threshold 0.90)

(g) randomized GS (threshold 0.95)

(h) PC (threshold 0.90)

(i) PC (threshold 0.95)

Figure 3.10: An example reconstruction of a Bayesian network of 25 nodes and 52 edges by different structure induction techniques, from 20,000 samples drawn from the distribution represented by the original network using logic sampling.

The plain GS network with threshold 0.90 (Fig. 3.10(d)) is able to identify 45 out of 52 edges in the original BN. One edge that was not present in the original is erroneously included. Out of the 45 correctly identified edges, 42 of them were correctly oriented. The same GS algorithm with a stricter threshold of 0.95 (Fig. 3.10(e)) identified fewer edges correctly (39 out of 52) and 36 of them were correctly directed. However no extra edge that was not present in the original BN was included.

The randomized GS performs similarly to the plain GS algorithm. The version utilizing threshold 0.90 identifies 45 out of 52 edges, 34 of which are correctly directed, while there are 2 superfluous edges. The version with threshold 0.95 identifies 43 edges, 33 of which are correctly directed, and there is 1 superfluous edge.

The PC algorithm with threshold 0.90 identified 46 out of 52 edges correctly, similar to the GS algorithm under the same threshold. It also similarly contained only one extra edge. However only 8 out of the 46 edges present in the resulting network were correctly oriented. This behavior was encountered repeatedly in almost all experiments we conducted. The reason for it is the failure of the faithfulness assumption on which the PC algorithm is based. For example, if the network contains locally the subgraph $\{B \leftarrow A \rightarrow C\}$, it might be the case that B and C are unconditionally independent at the confidence level 0.90 or 0.95. Such an event is assumed not to happen under faithfulness. In order for the PC algorithm to determine whether or not there is an edge $B - C$ it finds the smallest set that d-separates B from C. This would be the empty set in this example. In a later step, and assuming that the edges $A - B$ and $A - C$ were correctly identified, the PC algorithm will orient these edges as $B \rightarrow A$ and $C \rightarrow A$, due to the fact that A is not in the minimum separator set of B and C, and thus commit two directionality errors. In addition, more directionality errors may follow due to the direction propagation step of the PC algorithm that will occur at a later stage.

The BIC hill-climbing algorithm starting from an empty network is shown in Fig. 3.10(b). It is successful in discovering 50 out of 52 edges in the original network. However it also produces 5 extra edges, and only 41 out of the 50 correctly identified edges are correctly directed.

We also report in Fig. 3.10(c) the best BN (in terms of score) returned by the BIC hill-climbing algorithm using 10 random structures as starting points of the hill-climbing process. Similarly to the network in Fig. 3.10(b) it returns 50 out of 52 edges. However it also returns 17 extra edges not present in the original BN. It also correctly orients only 37 of the 50 edges correctly.

It is clear that working with large Markov blankets presents severe problems in terms of computational efficiency and data set size, since conditional probability tables of size exponential in the size of the Markov blanket are involved. Including too many nodes in the initial Markov blanket growing phase of the GS algorithm would handicap almost all subsequent steps, especially if the excess nodes are not removed during the shrinking phase. In addition, nodes incorrectly excluded during this phase also presents a problem since there is no opportunity to add them at a later step. A crucial parameter affecting both these phases is the threshold of the conditional independence test. Two thresholds are typically used in practice, 0.90 and 0.95.

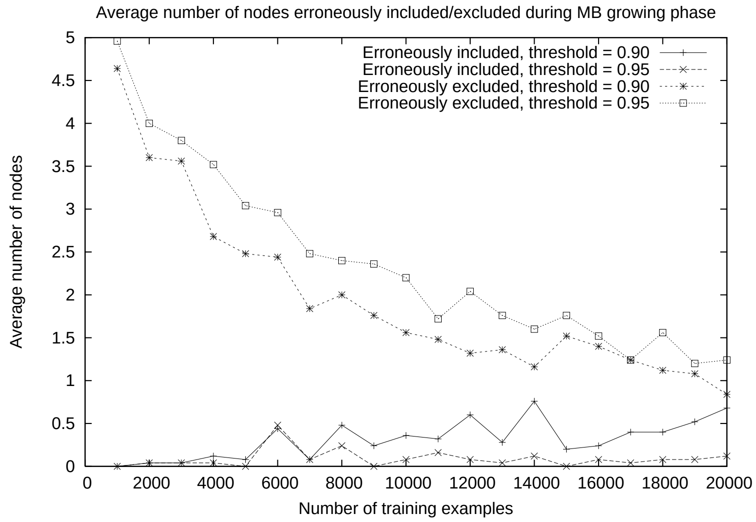


Figure 3.11: Number of nodes incorrectly included and incorrectly excluded during the Markov blanket growing phase. Results for a 5×5 rectangular network with branching factor 3 (in both directions, corresponding blanket size 9).

To get an idea of the significance of the use of each of these thresholds we plot in Fig. 3.11 the number of nodes of the Markov blanket incorrectly included and incorrectly excluded during the growing phase, averaged over all nodes in the domain. As expected, the number of nodes incorrectly included is less for the "stricter" threshold 0.95, that the threshold 0.90. On the other hand, also as expected, use of threshold 0.95 results in more missing nodes. The figure tends to favor the choice of 0.95, since the nodes incorrectly included are generally very low and the nodes in-excess fall rapidly with increasing sample size and are approximately equal to the number for threshold 0.90 at 20,000 data points. For these reasons we only use 0.95 in the subsequent figures.

Using confidence level 0.95, the likelihood of the training data set is much higher for the hill-climbing algorithms as can be seen in Fig. 3.12(a). We can observe that the PC and GS algorithms do similarly in terms of data likelihood and worse than the hill-climbing ones. This is also the case with the number of edges correctly identified, shown in Fig. 3.12(b), where the hill-climbing algorithms are also superior. However, this trend is reversed in Fig. 3.12(c), where we can observe that they suffer from an excessive inclusion of superfluous edges, compared to both GS and PC algorithms. The latter algorithms contain very few (frequently zero) extra edges, and can be characterized as more conservative in that respect. The large number of extra edges is consistent and seems to explain the superiority of the hill-climbing algorithms in term of data likelihood, since the addition of edges in a BN can only increase the data likelihood. Finally, in Fig. 3.12(d) we can see that the PC algorithm does poorly in terms of number of edges correctly directed, due to reasons mentioned above. We believe that a version of the PC algorithm that is more robust to possible failure of the faithfulness assumption that becomes competitive with GS is possible. At this point however,

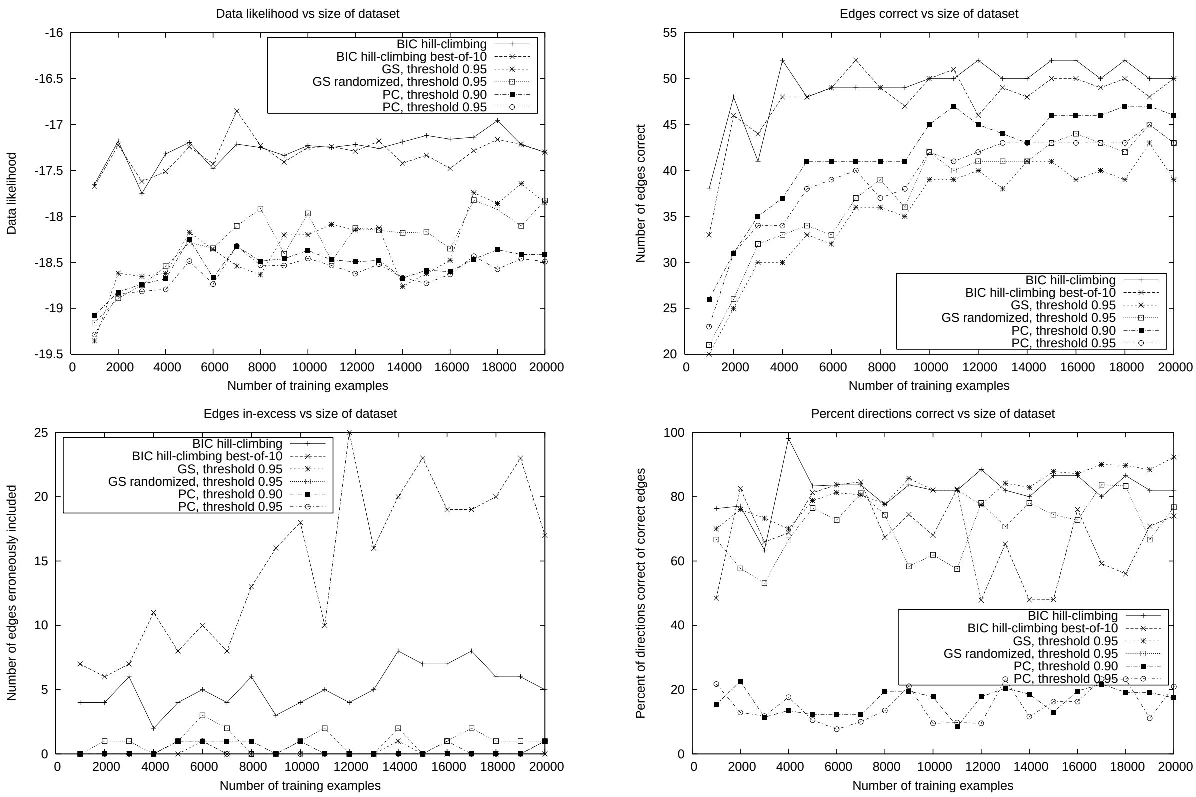


Figure 3.12: Data likelihood (**top-left**) and structure accuracy (**top-right** and **bottom**) results for the BIC hill-climbing, plain GS, randomized GS, and PC algorithms on a synthetic example, as a function of the number of training examples.

such an algorithm is not available.

Fig. 3.13 shows the effects of increasing the Markov blanket through an increasing branching factor. As expected, we see a dramatic (exponential) increase in execution time of the plain GS algorithm, though only a mild increase of the randomized version. The latter uses 100 conditional tests per decision, and its execution time increase is attributed to the (quadratic) increase in the number of decisions. Note that the edge percentages between the plain and the randomized version remain relatively close. The number of direction errors for the GS algorithm actually decreases due to the larger number of parents for each node (more “V” structures), which allows a greater number of opportunities to recover the directionality of an edge (using an increased number of tests).

3.4.2 Real-world Datasets

We used the following real-world datasets from the UC Irvine Machine Learning Repository (Murphy and Aha, 1994):

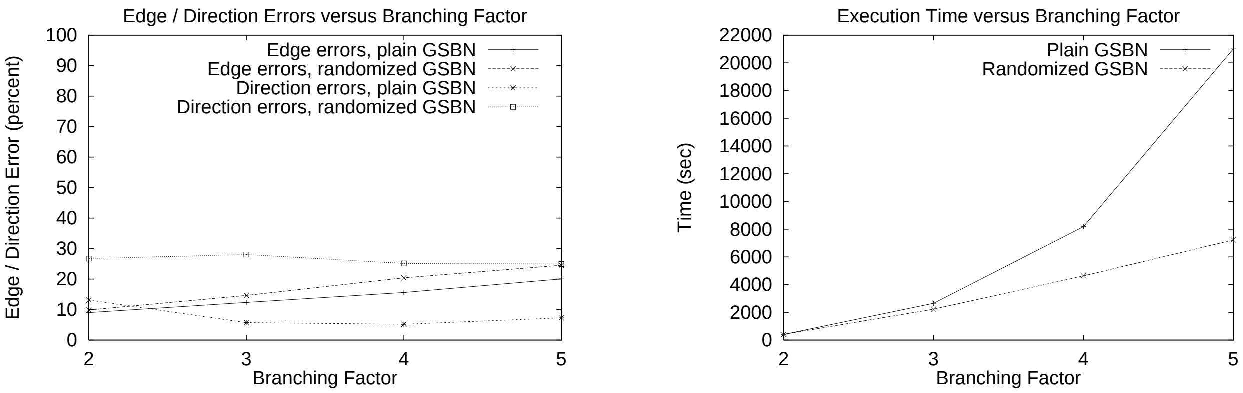


Figure 3.13: Results for a 5×5 rectangular network from which 10,000 samples were generated and used for reconstruction, versus increasing branching factor. On the left, errors are slowly increasing (as expected) or remain relatively constant. On the right, corresponding execution times are shown. While the plain GS algorithm execution time increases exponentially with branching factor, the randomized version has a much milder increase in execution time.

1. The ADULT dataset. This contains demographic information about individuals gathered from the Census Bureau database. The original dataset contained a mixture of discrete and continuous attributes, 16 in total. We used only the discrete attributes, which were 9. The dataset was split into a training set of size 32,562 and a test set of size 16,283 (a 2/3 to 1/3 split). After eliminating records with unknown attributes for the discrete attribute (for simplicity), we ended up using a training set of size 30,162 and a test set of size 15,060. The attributes we used were:

   (a) work class,

   (b) education,

   (c) marital status,

   (d) occupation,

   (e) relationship,

   (f) race,

   (g) sex,

   (h) native country.

2. The SPLICE dataset. This dataset contains DNA sequences from the Genbank-64.1 database and identifies extron-intron (EI) and intron-extron (IE) boundary positions (splice junctions) and 30 nucleotides (A, C, T, G base pairs) on either side surrounding those sites. In certain positions where there is uncertainty, additional "virtual nucleotides" D (A or G or T), S (C or G) and R (A or G) are included in the sequences. We created one record from each EI or IE example containing 61 features. We also split the set to training (2,128 records) and test sets (1,047 records). The attributes used are:

(1) the Class label (EI or IE),

(2)–(61) the nucleotide (real or virtual) at position -30 to position $+30$ around the splice junction.

3. The CAR dataset. This contains information about automobiles meant to help a customer evaluate a particular car. There were 1728 records of 7 attributes each. There was no test set. We evaluated the different algorithms using the same dataset for training and testing. The reason for this is to test the overfitting avoidance features of the algorithms, which are particularly severe in such a small data set since small data sets are frequent in practice. In these cases a test data set might be a luxury that the researcher might not be able to afford. The attributes (all discrete) were:

   (a) buying price,

   (b) maintenance cost,

   (c) number of doors,

   (d) capacity in number of persons to carry,

   (e) size of the luggage boot (trunk),

   (f) estimated safety of the car,

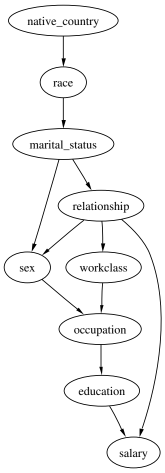
   (g) the class of the car (acceptable or unacceptable).

The ADULT Real-world Dataset

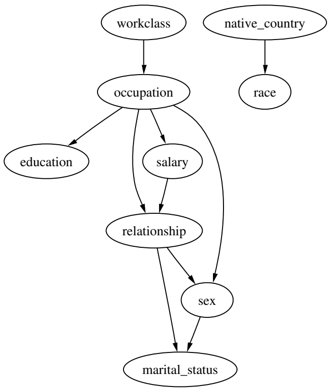
The resulting network structures of our experiments on the ADULT dataset are displayed in Fig. 3.14. Fig. 3.14(a) shows the output networks using standard hill-climbing using the BIC score starting from an empty network (no edges) while Fig. 3.14(b) shows the best network (in terms of maximum BIC score) after 10 restarts using random networks as starting points.

Note that the output of the GS algorithm for thresholds 0.90 and 0.95 are similar (Fig. 3.14(e) and (f)), at least as far as undirected connectivity is concerned; there is only one undirected arc missing from the BN created using 0.95 confidence for the χ^2 test compared to the one using a 0.90 confidence. Restarting BIC hill-climbing is not as stable: there are 5 undirected arc differences between the BN created starting from an empty network and the best-of-10 restarts one.

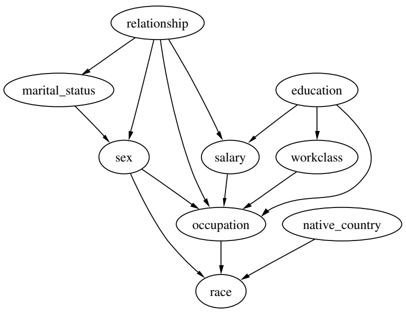
Regarding edge directions, the results are not as good for the GS algorithm. There are 7 directionality differences on the edges common to both BNs produced by the GS algorithm on the two different thresholds. These seem to be due to the density of the resulting graph, which produces fewer opportunities for tests at Step 3 of the GS algorithm (Fig. 3.7), since two neighbors of X that are parents may be connected by an edge (each belonging to the neighborhood of the other) and therefore excluded from being used in Step 3.



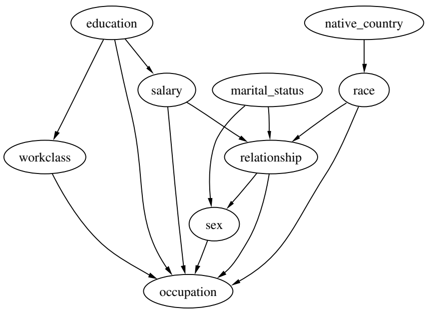
(a) BIC hill-climbing (no restarts)



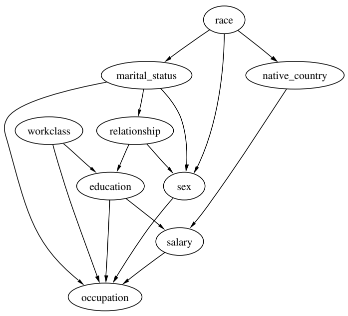
(b) BIC hill-climbing (best of 10 restarts)



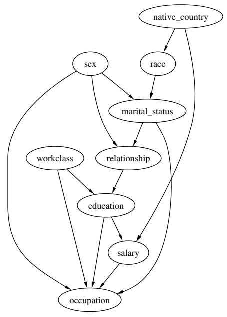
(c) PC (confidence threshold 0.90)



(d) PC (confidence threshold 0.95)



(e) GS (confidence threshold 0.90)



(f) GS (confidence threshold 0.95)

Figure 3.14: Resulting BNs from the ADULT real-world dataset.

Data Likelihood of Test Dataset			
Hill-climbing BIC (no restarts)	Hill-climbing BIC (best-of-10 restarts)	PC (threshold 0.90)	PC (threshold 0.95)
-12.3216	-12.0913	-12.45322	-12.43684

Data Likelihood of Test Dataset		
GS (threshold 0.90)	GS (threshold 0.95)	Independent BN
-11.15419	-11.97039	-13.61859

Figure 3.15: Data likelihood figures for the ADULT test dataset for each BN.

The BIC approach was less robust (more sensitive to starting point of the structure search) in this dataset with regard to directionality error. There are 13 directionality differences between the no-restarts and the best-of-10 restarts BN produced by hill-climbing on the BIC score surface.

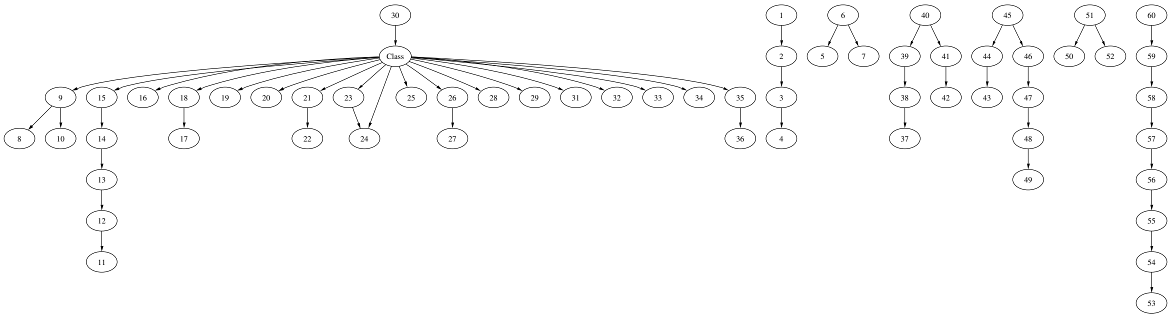
The data likelihood of the resulting networks are shown in the table of Fig. 3.15. In this we calculate the likelihood of a test dataset of 15,060 examples that was not used for training any of the resulting BNs. From the numbers in the figure we see that the best approach is GS with threshold of 0.90, followed by GS with threshold of 0.95. For comparison purposes we include the "strawman" network that contains no edges in the rightmost column, under "Independent BN," which performs the worst.

The SPLICE Real-world Dataset

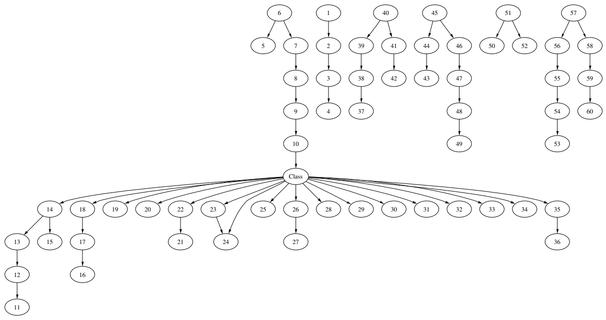
Each record in the SPLICE dataset contains segments of 60 nucleotides or "base-pairs" occurring in the neighborhood of an intron-exon (IE or "acceptor" site) or exon-intron (EI or "donor" site) boundary. In particular provided are the 30 base-pairs occurring on the left and the 30 ones on the right of the boundary. Recognizing these boundaries is important because they help identify "splice junctions," which are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms. The portions that are retained are the exons while the introns are discarded during this splicing process.

We processed each of these records in as simple a fashion as possible: we created a variable for each location (-30 to $+30$) plus an additional "class" binary variable that takes the values EI and IE. We split the original dataset of 3,190 records into a training set of 2,128 records and a test set of 1,047 records.

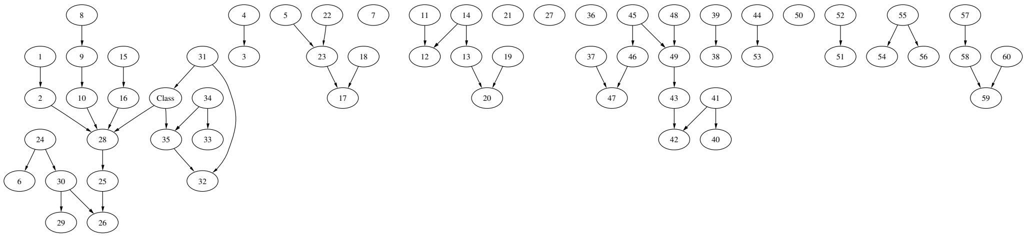
Due to the large number of attributes in this domain (61), and to the extremely large resulting structure returned by the PC algorithm (for both thresholds, 0.90 and 0.95), no network was available for this algorithm. The network structure returned by the PC algorithm suggested more than 30 parents to the "Class" variable, which would require more memory for the conditional probability tables (and processing power to produce and access them sequentially) that there is available in almost any computer today.



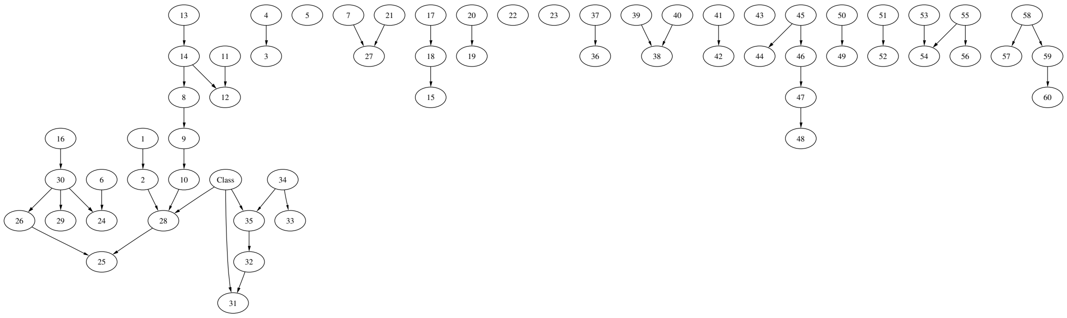
(a) BIC hill-climbing (no restarts)



(b) BIC hill-climbing (best of 10 restarts)



(c) GS (confidence threshold 0.90)



(d) GS (confidence threshold 0.95)

Figure 3.16: Resulting BNs from the SPLICE real-world dataset.

Data Likelihood of Test Dataset				
Hill-climbing BIC (no restarts)	Hill-climbing BIC (best-of-10 restarts)	GS (threshold 0.90)	GS (threshold 0.95)	Independent
-118.1442	-117.1381	-118.5051	-117.7139	-120.0293

Figure 3.17: Data likelihood figures for the SPLICE test dataset for each BN.

As in the ADULT dataset, we applied the BIC hill-climbing algorithms with and without the best-of-10 restarts. We also applied GS with thresholds 0.90 and 0.95. We display the resulting structures in Fig. 3.16. As we can see most of the networks are sparse, indicating the absence of strong dependences among the different base-pairs at different locations. However, the most important variable is the one indicating the class of the boundary. We see that variable tends to be connected to neighboring base-pairs (*i.e.* pairs around variable "30" which occurs at the boundary) in both networks produced by the GS algorithm. This characteristic is absent from the networks produced by hill-climbing on the BIC score.

The data likelihoods on a separate test set that was not used in training is shown in Fig. 3.17. There we see that, not surprisingly, the best-of-10 restarts BIC hill-climbing procedure performs the best. It is followed by the GS procedure with threshold 0.95, BIC hill-climbing with no restarts and GS with threshold 0.90.

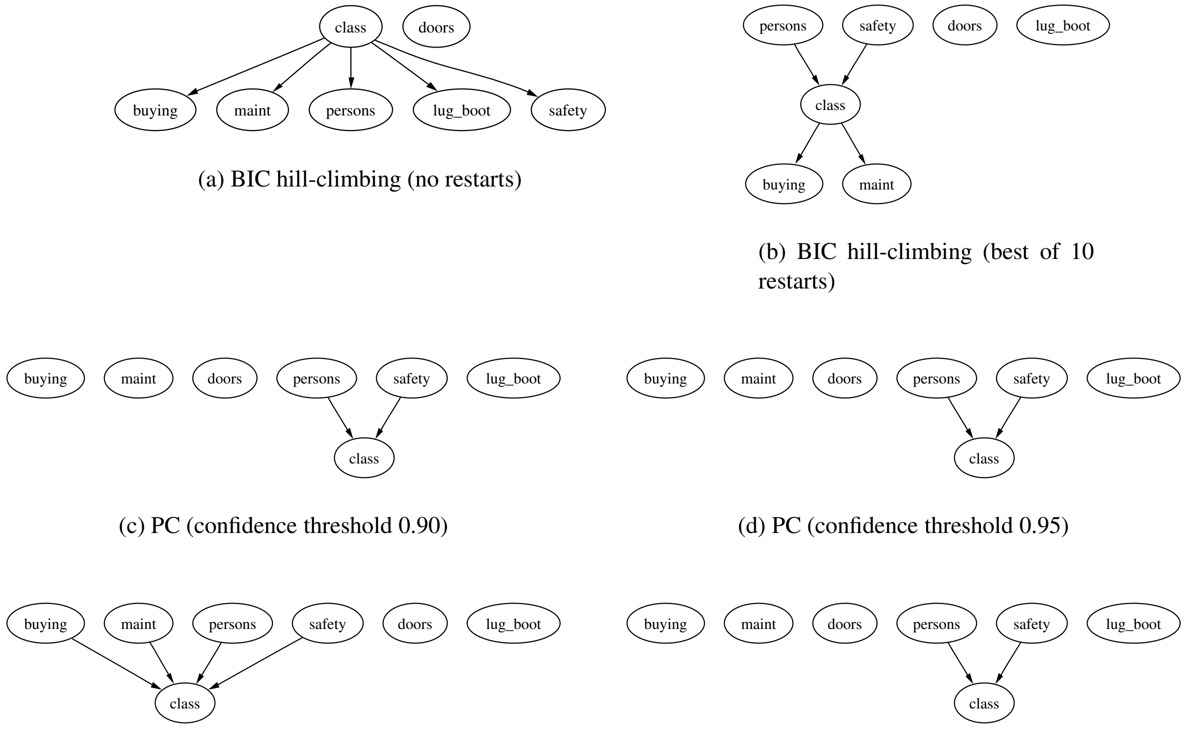
The CAR Real-world Dataset

The CAR dataset is not exactly a "real-world" model in the sense that it contains instances drawn from a known hierarchical model that was used to evaluate cars. The actual structure of the model is the following:

CAR (car acceptability)

- **PRICE** (overall price)
  - **buying** : buying price
  - **maint** : maintenance cost
- **TECH** (technical characteristics)
  - **doors** : number of doors
  - **persons** : capacity in terms of persons to carry
  - **lug_boot** : the size of luggage boot
- **safety** : estimated safety of the car

The variables involved in the model were: "CAR" (output attribute), and "buying," "maint," "doors," "persons," "lug_boot," and "safety" (input attributes). The remaining quantities above ("PRICE," and "TECH")



(a) BIC hill-climbing (no restarts)

(b) BIC hill-climbing (best of 10 restarts)

(c) PC (confidence threshold 0.90)

(d) PC (confidence threshold 0.95)

(e) GS (confidence threshold 0.90)

(f) GS (confidence threshold 0.95)

Figure 3.18: Resulting BNs from the CAR dataset, that was sampled from a real-world model.

are intermediate quantities (internal nodes of the decision tree). The decision process maps to an actual causal network that is an "upside-down tree," with the input attributes at the top and the class node (CAR) at the bottom.

The results of the different algorithms are shown in Fig. 3.18. As we can see, the GS algorithm with threshold 0.90 does the best job of capturing the "upside-down tree" structure of the model from which the dataset originated, although some edges are missing. The same algorithm with threshold 0.95 misses some more edges due to the greater stringency of the test of independence and the small number of examples in the dataset. The BIC hill-climbing algorithm with no restarts produces a model that actually contains much fewer parameters but does not capture the causal structure of the domain. We conjecture that the reason for this is the fact that there are few examples in the dataset and therefore the BIC score approximation (Eq. (2.1)) does not hold well. The BIC hill-climbing with restarts does better, but still does not produce an inverted tree.

Due to the small number of examples, we did not split the dataset into a training and a test set. Another reason for this is our desire to experimentally check the feature of the BIC score *i.e.* the fact that it automat-

Data Likelihood of Test Dataset			
Hill-climbing BIC (no restarts)	Hill-climbing BIC (best-of-10 restarts)	PC (threshold 0.90)	PC (threshold 0.95)
−11.05425	−11.03446	−11.06901	−11.06901

Data Likelihood of Test Dataset		
GS (threshold 0.90)	GS (threshold 0.95)	Independent BN
−11.18333	−11.06901	−11.44279

Figure 3.19: Data likelihood figures for the CAR *training* dataset for each BN.

ically avoids overfitting (due to the penalty term—see Eq. (2.1)) and thus does not require a cross-validation (test) dataset.

As we can see in the data likelihood figures in Fig. 3.19 the BIC algorithms do a good job in capturing the probability distribution of the dataset, with the best-of-10 restarts version slightly better. The GS algorithms are similar though, and slightly behind. This is a clear example of cases where the highest scoring model is not clearly the most desirable one, in the sense of capturing the underlying causal structure. The GS algorithm, esp. the version utilizing the 0.90 threshold, does that best from all approaches we tried. Incidentally, we note that the attributes "buying," "maint," "persons," and "safety" are indeed pairwise independent, as deemed by the χ^2-test, and do become dependent upon conditioning on the "class" variable. This is to be expected since the upside-down tree structure was the one used to produce the dataset in reality.

3.5 Discussion and Comparison to Other Approaches

The algorithms presented here have similarities to the SGS, IC and PC algorithms (see Section 2.7.3), especially the plain GS. The latter can be viewed as a version of the SGS algorithm with the conditioning sets restricted to the Markov blanket of each node. The SGS algorithm proceeds in a fashion similar to the GS algorithm; for example, in order to determine the existence of an edge between two nodes X and Y, the SGS algorithm employs independence tests conditioned on every subset of $\mathcal{U} - \{X, Y\}$. This is clearly wasteful for sparse graphs where many variables may be irrelevant, and it also places unnecessarily requirements on the size of the data set that is needed for reliable structure discovery. The GS algorithm circumvents both problems by employing a preliminary step in which it discovers the local structure around each node. Concerning the recovery of the directionality of the links, the approach of the SGS algorithm is analogous to a global version of step 3 of the GS algorithm. It is thus inefficient for the same reasons.

Performance-wise, it is more fair to compare the GS algorithm with an algorithm that heeds local structure such as the the PC algorithm (Spirtes et al., 1993). The PC algorithm is a much more efficient algorithm than

SGS and it may be employed in practice on problems involving a significantly larger number of variables. In short, its operation is as follows. Initially a completely connected graph is constructed. This graph is then gradually "thinned" using conditional independence tests of increasing order (conditioning set size), starting from the empty set. For an ordered pair of variables X and Y connected with a hypothesized edge, the algorithm makes a number of tests, trying all conditioning sets of a certain size from the current set of direct neighbors of X. In a sparsely connected original network, it is expected that many links will be eliminated early, reducing the sizes of the direct neighbors sets and improving the running times of subsequent steps. Each conditioning set that successfully separates two variables is recorded and used in a later phase where the directions of the remaining links are determined in a fashion similar to the SGS (and the GS) algorithm.

Spirtes et al. (1993) present an informal worst-case analysis of the PC algorithm, which indicates that the number of independence tests used are $O(n^{k+1})$, where k is the maximum degree of a node of the original Bayesian network (in our notation, $k = u + d$, the sum of the maximum number of parents u and children d of any node). While both algorithms have a factor that is exponential in k, which most likely is unavoidable, the factor in GS is 2^b, where $b = k + u$ in the worst case. Therefore, although the PC algorithm is indeed efficient (*i.e.* polynomial in the number of tests) under the assumption of a bounded neighborhood (k bounded by a constant), unfortunately the order of the polynomial depends on k, while the GS algorithm does not. For a given k (and therefore b), the GS algorithm has an asymptotic behavior with respect to n of $O(n^2)$, superior to the one achieved by PC algorithm. This is likely to make a difference in large graphs that are sparsely connected.

There is another family of algorithms for structure recovery, as mentioned in Section 2.7.2. Until very recently the problem with greedy search in the space of DAG structures was that it was not guaranteed to return a network that is faithful in terms of independence relations, nor one whose structure is "close" to the original one; their guarantee was to return a network for which the scoring function, which is often related to the likelihood of the input data, is at a local maximum. (Such networks may be usefully employed to compute the probability of future data assuming they are drawn from the same distribution.) Therefore it is conceivable that an application of one of these algorithms be used to "fine-tune" the network returned by the GS algorithm, possibly producing a structure similar to the original one that is also close to the global maximum of the data likelihood. Very recently Chickering (2002) made a significant contribution by providing an algorithm that can return an equivalence class of BNs that is optimal along certain dimensions using a greedy search in the space of equivalence classes, in the limit of an input dataset of infinite size.

Chapter 4

Testing for Independence

In this chapter we focus on an integral component of discovering the structure of a Bayesian network, namely developing a probabilistic conditional independence test for use in causal discovery. There exist several approaches for BN structure discovery in domains with categorical variables (Spirtes et al., 1993; Pearl, 2nd Ed., 1997; Lam and Bacchus, 1994a; Suzuki, 1996). However there are few for continuous or hybrid domains (Friedman et al., 1998; Monti and Cooper, 1998a), mostly employing score-maximization approaches for structure induction. In contrast to this, the constraint-based family (Pearl and Verma, 1991; Spirtes et al., 1993; Margaritis and Thrun, 2000) employs conditional independence tests, and has guarantees of inducing the correct causal structure (under assumptions). However, to date there is no general approach to constraint-based structure induction for continuous or hybrid domains, mainly due to the difficulty of testing for independence in the general case. Our approach attempts to remedy that, by developing a statistical independence test for continuous variables that places no constraints on the probability distribution of the continuous variables of the domain. In this chapter we propose an algorithm which works by eliciting information from continuous data at several resolutions and combining it into a single *probabilistic measure of independence*. For this purpose, the examination of many resolutions is necessary. An example that helps explain why that is the case is shown in Fig. 4.1. Two scatterplots are shown, together with their corresponding discretizations at 3×3 resolution. The histograms for the two data sets appear similar even though the independence of the axes variables X and Y is not. The data on the left show a strong dependence between X and Y while the data on the right plot are independent by construction. In this case, finer resolution histograms need to be examined in order to determine that dependence. This observation is formalized in Section 4.1 below.

At the top level, our approach is formulated as a Bayesian model selection problem, assigning the class of independent distributions a prior probability. This formulation sidesteps the fact that the class of independent distributions lies on a zero-support submanifold in the domain of distribution functions. This is one of the key advantages of the Bayesian approach that our method relies on.

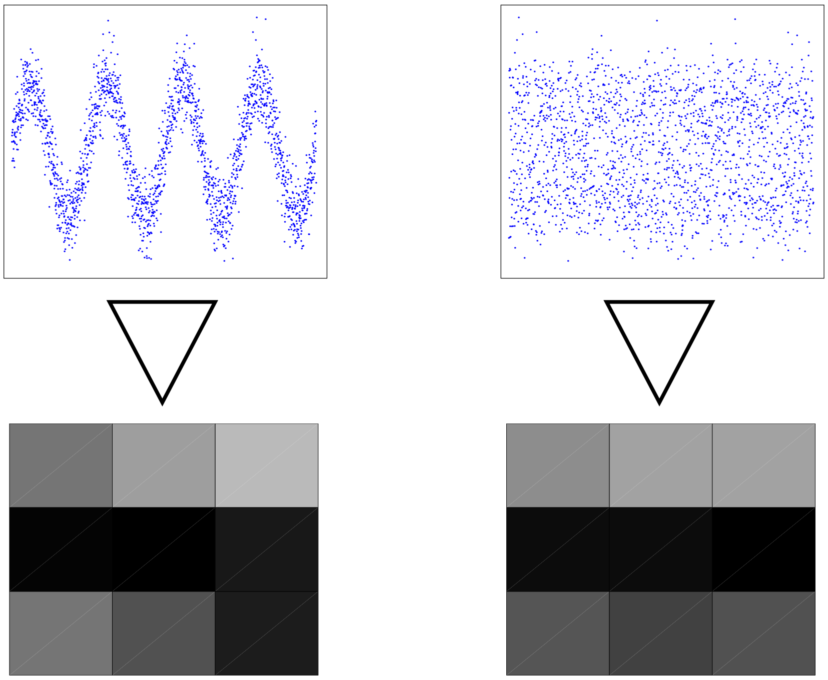


Figure 4.1: Two very different data sets have similar 3×3 histograms. **Left:** Data strongly dependent. **Right:** Data independent by construction.

In the next section we present out approach in detail. Our discussion assumes that all variables are continuous. Application to domains that contain ordinal discrete and categorical variables as well as continuous is straightforward and is only briefly mentioned in Chapter 6.

4.1 Method Description

Let us assume two continuous variables X and Y whose independence we are interested in measuring. If the marginal probability distribution function (pdf) of each is $\Pr(X)$ and $\Pr(Y)$ respectively, and the joint pdf is $\Pr(X,Y)$, then

> **Definition (independence):** X and Y are independent iff $\Pr(X,Y) = \Pr(X)\Pr(Y)$, for all values x and y in the domains of X and Y, respectively.

As we mentioned above, our method tests for independence at many resolutions. It does so by discretizing the multidimensional space of the variables involved in the test at several resolutions. In this section we present our approach in two stages. In order to explain our method more effectively, we first describe the simple case for testing independence at a single, fixed resolution.

4.1.1 Single resolution, fixed grid test

In this section we are given a $I \times J$ table of counts that represent the discretized scatterplot of variables X and Y, and we develop an *exact* test of independence. Such a test has the potential of being used in situations where data are sparse. It can also be used at fine resolutions, a task that will be necessary in the multi-resolution test, described later in Section 4.1.2. We first present the case of an unconditional test of independence $(X \perp Y)$ and propose an extension of the method to a conditional test $(X \perp Y \mid \mathbf{C})$ for a given resolution $I \times J \times C_1 \times C_2 \times \cdots \times C_{|\mathbf{C}|}$ near the end of this section.

We assume that the counts of the table, $c_1, \ldots, c_K, K \stackrel{\text{def}}{=} IJ$, follow a multinomial distribution. The choice of a multinomial is in a sense the most "unbiased" one because it does not make any implicit assumptions about any interaction between adjacent bins (sometimes called "smoothing"). We denote the resolution as $R \stackrel{\text{def}}{=} I \times J$, the set of grid boundaries along the axes as $\mathbf{B}_R$, and the probability of each bin as $p_k, k = 1, 2, \ldots, K$ (for brevity, in the following we denote such a set of numbers as $p_{1\cdots K}$). The probability of the data set $\mathcal{D}$ (the data likelihood) is the likelihood of the bin counts, namely

$$\begin{aligned}
\Pr(\mathcal{D} \mid p_{1\cdots K}, \mathbf{B}_R, R) &= \Pr(c_{1\cdots K} \mid p_{1\cdots K}, \mathbf{B}_R, R) \\
&= N! \prod_{k=1}^{K} \frac{p_k^{c_k}}{c_k!} \qquad (4.1)
\end{aligned}$$

where $N = |\mathcal{D}|$ is the size of our data set. (For brevity, in the remainder of this section, we omit $\mathbf{B}_R$ and R from all probability terms that appear, but assume that they implicitly condition on them.) Since the parameters $p_{1\cdots K}$ are unknown, we adopt a Bayesian approach: we use a prior distribution $\Pr(p_{1\cdots K})$ over them. Given such a distribution the data likelihood is the average over all possible parameter values, weighted by their probability:

$$\Pr(\mathcal{D}) = \int \Pr(\mathcal{D} \mid p_{1\cdots K}) \Pr(p_{1\cdots K}) \mathrm{d}p_{1\cdots K}. \qquad (4.2)$$

The most commonly used prior distribution for multinomial parameters is the Dirichlet:

$$\Pr(p_{1\cdots K}) = \Gamma(\gamma) \prod_{k=1}^{K} \frac{p_k^{\gamma_k - 1}}{\Gamma(\gamma_k)},$$

where $\gamma = \sum_{k=1}^{K} \gamma_k$ and $\Gamma(x)$ is the gamma function. The positive numbers $\gamma_{1\cdots K}$ of this distribution are its hyperparameters.[1] When $\gamma_k = 1$ for all k, the Dirichlet distribution is uniform.

The choice of a Dirichlet prior has certain computational advantages. Since it is conjugate prior to the multinomial, the posterior distribution of the parameters is also Dirichlet (albeit with different parameters).

[1] The hyperparameters can be thought of as "virtual samples."

This enables us to compute the integral (4.2) analytically:

$$\Pr(\mathcal{D}) = \frac{\Gamma(\gamma)}{\Gamma(\gamma+N)} \prod_{k=1}^{K} \frac{\Gamma(\gamma_k + c_k)}{\Gamma(\gamma_k)}. \tag{4.3}$$

To test independence, we assume that our data have been produced by one of two classes of models, one representing independence and one that does not. The former one contains $I+J$ parameters for the marginal probabilities of the rows and columns of the $I \times J$ table. We call this model class M_I, and denote its prior probability as $\wp \stackrel{\text{def}}{=} \Pr(M_I)$. The fully dependent model class, denoted $M_{\neg I}$, has prior probability $\Pr(M_{\neg I}) = 1 - \wp$. The posterior probability of independence, $\Pr(M_I \mid \mathcal{D})$, is

$$\Pr(M_I \mid \mathcal{D}) = \frac{\Pr(\mathcal{D} \mid M_I)\Pr(M_I)}{\Pr(\mathcal{D})}$$

by Bayes' theorem. Since

$$\Pr(\mathcal{D}) = \Pr(\mathcal{D} \mid M_I)\Pr(M_I) + \Pr(\mathcal{D} \mid M_{\neg I})\Pr(M_{\neg I})$$

we get

$$\Pr(M_I \mid \mathcal{D}) = 1 \Big/ \left[1 + \frac{(1-\wp)}{\wp} \frac{\Pr(\mathcal{D} \mid M_{\neg I})}{\Pr(\mathcal{D} \mid M_I)}\right]. \tag{4.4}$$

At resolution R, the term $\Pr(\mathcal{D} \mid M_{\neg I})$ of the fully dependent model that contains $K = IJ$ parameters is given by (4.3), *i.e.*

$$\begin{aligned} \Pr(\mathcal{D} \mid M_{\neg I}) &= \frac{\Gamma(\gamma)}{\Gamma(\gamma+N)} \prod_{k=1}^{K} \frac{\Gamma(\gamma_k + c_k)}{\Gamma(\gamma_k)} \qquad (4.5) \\ &\stackrel{\text{def}}{=} \Upsilon(\mathbf{c}_K, \gamma_K). \end{aligned}$$

For the independent model we assume two multinomial distributions, one each along the X and Y axes, that contain J and I parameters, respectively. These correspond to the marginal probabilities along the axes. We denote the marginal count at column j as c_{+j} and at row i as c_{i+}. The marginal probabilities (which are unknown) are denoted as $q_{1\ldots I}$ with prior a Dirichlet with hyperparameters $\alpha_{1\ldots I}$, and $r_{1\ldots J}$ with hyperparameters $\beta_{1\ldots J}$. The probability of bin (i,j) under M_I is $q_i r_j$. The data likelihood is computed in a manner analogous to Eq. (4.3):

$$\begin{aligned} \Pr(\mathcal{D} \mid M_I) &= \left(\frac{\Gamma(\alpha)}{\Gamma(\alpha+N)} \prod_{i=1}^{I} \frac{\Gamma(\alpha_i + c_{i+})}{\Gamma(\alpha_i)}\right) \left(\frac{\Gamma(\beta)}{\Gamma(\beta+N)} \prod_{j=1}^{J} \frac{\Gamma(\beta_j + c_{+j})}{\Gamma(\beta_j)}\right) \\ &\stackrel{\text{def}}{=} \Upsilon(\mathbf{c}_{I+}, \alpha_I)\Upsilon(\mathbf{c}_{+J}, \beta_J) \qquad (4.6) \end{aligned}$$

again with $\alpha = \sum \alpha_i$ and $\beta = \sum \beta_j$. Combining Eq. (4.4), (4.5) and (4.6), we arrive at our final formula of the posterior probability of independence at resolution R:

$$\Pr(M_I \mid \mathcal{D}) = 1 \Bigg/ \left[1 + \frac{1-\wp}{\wp} \frac{\Upsilon(\mathbf{c}_K, \gamma_K)}{\Upsilon(\mathbf{c}_{I+}, \alpha_I)\Upsilon(\mathbf{c}_{+J}, \beta_J)} \right]. \tag{4.7}$$

Eq. (4.7) refers to a fixed grid at resolution $I \times J$, not necessarily regular, with a given boundary set $\mathbf{B}_{I\times J}$.

We note that the above test applies to categorical variables only. Random rearrangement of the rows and/or columns of the table will not affect the joint or the marginal counts and thus not affect $\Pr(M_I \mid \mathcal{D})$. However the resulting table counts appear more random, since the bin positions have been randomly changed, including the points that are contained in each of them. A more powerful test that takes into account the ordering and relative position of the points is described in the next section. This involves "sweeping" the discretization boundaries across the XY plane and computing a Bayesian average of the results.

Lastly, we briefly address conditional independence: a conditional test $(X \perp Y \mid \mathbf{C})$ at a fixed resolution $I \times J \times C_1 \times C_2 \times \cdots \times C_{|\mathbf{C}|}$ is simply the product of probabilities of independence of X and Y for each $I \times J$ "slice" from $(1,1,\ldots,1)$ to $(C_1, C_2, \ldots, C_{|\mathbf{C}|})$. This is necessary because a conditional independence statement is equivalent to independence for all values of the conditioning set.

In the next section we describe a test that takes into account multiple resolutions.

4.1.2 Multi-resolution Test

As we mentioned in the introduction, to estimate the posterior probability of independence we need to examine our data set at multiple resolutions. We first make the observation that if X and Y were independent, then they would appear independent when discretized at every resolution $R = I \times J$. Therefore, it seems that testing for independence then would require checking independence at all resolutions *i.e.*

$$\Pr(M_I \mid \mathcal{D}) = \Pr(M_I^{R_1} \wedge M_I^{R_2} \wedge \cdots \mid \mathcal{D}),$$

where $\Pr(M_I^R)$ denotes the statement that X and Y appear independent at resolution R. The conjunction goes over all resolutions $R = I \times J$. The number of such resolutions is countably infinite. However, the following crucial observation about the structure of the space of resolutions facilitates our computation:

> **Observation:** If X and Y are independent at resolution $2R \stackrel{\text{def}}{=} 2I \times 2J$, they are independent at resolution $R \stackrel{\text{def}}{=} I \times J$. In other words, $\Pr(M_I^R \mid M_I^{2R}) = 1$ for all R.

It is straightforward to prove the validity of the above—it can be demonstrated by using a $2I \times 2J$ grid where each bin probability is the product of the corresponding marginal bins, and by combining adjacent bins in 2×2 groups.

Given the above observation, we have

$$\begin{aligned}\Pr(M_I^R \wedge M_I^{2R}) &= \Pr(M_I^R \mid M_I^{2R}) \Pr(M_I^{2R}) \\ &= \Pr(M_I^{2R}).\end{aligned}$$

This implies that in order to ascertain independence, we do not need take a conjunction over all resolutions; instead, we should examine our data set at a resolution as fine as possible. Unfortunately, because our data set size is finite, we can only obtain histograms of up to a finite resolution. Estimating the posterior probability of independence at a fine, limiting resolution R_{max} is therefore an important part of our algorithm. To do that, we again use a Bayesian approach and average over the possible choices, weighed by their posterior:

$$\Pr(M_I \mid R_{max}, \mathcal{D}) = \int \Pr(M_I \mid \mathbf{B}, R_{max}, \mathcal{D}) \Pr(\mathbf{B} \mid R_{max}, \mathcal{D}) \mathrm{d}\mathbf{B}.$$

The integral runs over all possible sets of discretization grid boundaries $\mathbf{B}$ at resolution R_{max} *i.e.* $\mathbf{B} \stackrel{\text{def}}{=} \mathbf{B}_{R_{max}}$. The Bayesian approach which we follow here, besides making the smallest number of unwarranted decisions, also possesses certain other advantages in our case; most notably it minimizes the possibility of spurious dependencies that may occur due to an unfortunate choice of boundary placement.

To compute the above integral we should ideally average over all possible histogram boundary placements along the X and Y axes. Lacking any other information, we assume a uniform prior distribution $\Pr(\mathbf{B} \mid R_{max})$ over grid boundary placement. Although theoretically the Bayesian integral runs over an infinity of possible such placements, given our data set we need only compute a finite number of them since many produce the same 2D histogram for our data. More specifically, we need only attempt boundary placements at each of the $(N-1)$ midpoints between successive data points along the X and Y axes, resulting in $(N-1)^2$ possible positions for each XY boundary pair. The posterior $\Pr(\mathbf{B} \mid R, \mathcal{D})$ (called w' in the algorithm pseudocode in Fig. 4.2) which we adopt for each such placement is one that uses the fraction of the 2D area between the two successive points along the X and Y axes, relative to the span of the entire set of data points in the XY plane.

Our algorithm for calculating the posterior probability of independence of X and Y, encompassing all the above comments, is shown in pseudocode in Fig. 4.2. It begins with a coarse 2×2 grid and successively refines it by adding carefully selected boundaries along the axes. At each step, the set of boundaries we admit—either an XY pair or a set containing one boundary for each variable in the conditioning set—is the one that increases the probability of dependence the most. From each new, (possibly) irregular grid, we compute its posterior probability of dependence using Bayesian averaging over the possible single-boundary choices (using Eq. (4.7) for each resulting grid) and we either iterate or stop.

Our algorithm is typically used in situations where a binary dependent/independent decision is required. In this case one can compare its output $\Pr(M_I \mid \mathcal{D})$ to the prior probability of independence $\wp$ and decide

To compute $\Pr(M_I \mid \mathcal{D})$:

```
B_save ← ∅
p_max ← 0
t ← 1
do:
    B ← B_save
    q_max ← 0
    for x ∈ { midpoints along X axis }
        for y ∈ { midpoints along Y axis }
            B' ← B ∪ {(x,y)}
            w' ← Pr(B' | R)
            p(x,y) ← [1 − Pr(M_I | B', R, D)] × w'
            if p(x,y) > q_max then
                q_max ← p(x,y)
                B_save ← B'
    p(t) ← Σ_{x,y} p(x,y)
    if p(t) > p_max
        p_max ← p(t)
    t ← t + 1
while the standard deviation of the members of the p(t) array is large
return 1 − p_max
```

Figure 4.2: Algorithm for computing the posterior probability of independence using an irregular multiresolution grid.

"independent" if and only if $\Pr(M_I \mid \mathcal{D}) > \wp$. In our experiments we used $\wp = 0.5$.

To see why the algorithm attempts to maximize the posterior probability of dependence we need to examine another implication of the observation made in the beginning of this section:

$$\begin{aligned}
\Pr(M_I^R) &= \Pr(M_I^R \wedge M_I^{2R}) + \Pr(M_I^R \wedge \neg M_I^{2R}) \\
&= \Pr(M_I^R \mid M_I^{2R}) \Pr(M_I^{2R}) + \Pr(M_I^R \mid \neg M_I^{2R}) \Pr(\neg M_I^{2R}) \\
&= \Pr(M_I^{2R}) + \Pr(M_I^R \mid \neg M_I^{2R}) \Pr(\neg M_I^{2R}) \\
&\geq \Pr(M_I^{2R}).
\end{aligned}$$

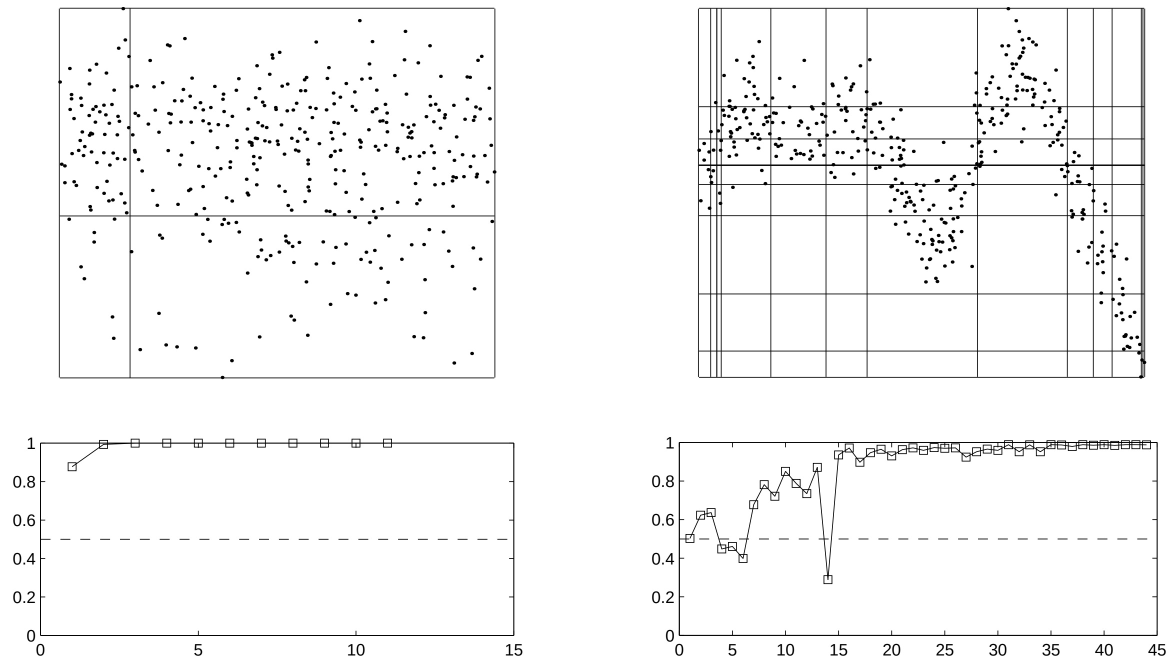
This implies that the probability of independence decreases or remains the same as we examine our data set at increasingly fine resolutions, or, equivalently, that the dependence probability is non-decreasing. Equality occurs in the case of independence at all resolutions. One can also verify intuitively that the above statement is true: imagine for example the limiting case where resolution tends to infinity for an ideal scenario where we have an infinite data set at our disposal. In this case, the only possibility for the independence posterior probability to remain constant at 1 is when the data set is truly independent at every resolution. Otherwise any discrepancies of the posterior probability of each bin from the product of the corresponding marginals will eventually emerge as we refine our discretization, causing the posterior independence probability to decline.

Combining the above observation with the previous one concerning the need to examine our data at as fine resolution as possible, we see that we need to maximize the posterior of dependence in an effort to discern dependence, as we progressively increase the effective resolution during a run of our algorithm. One concern however is that our data may appear increasingly artificially dependent at finer resolutions due to their increasing sparseness. Fortunately, this issue is automatically addressed by the fact that our approach is Bayesian: the number of parameters for the dependent model increases quadratically with the resolution, while the number of parameters of the independent one only linearly. Because the integral over all possible parameter values must evaluate to 1, the prior probability of any single parameter choice goes to 0 much faster for the dependent model at high resolutions than a choice of parameters for the independent model. Therefore, at high resolutions, the posterior of the independent model eventually "wins," in the sense of having a larger posterior probability. This helps us estimate the finest resolution to use: it is the one that returns a minimum for the posterior probability of independence. That posterior is "U-shaped," starting at a possibly high value at resolution 2×2, decreasing and then tending to 1 at high resolutions.

The above statement about the non-increasing probability of independence as we add boundaries refers to XY boundaries only. Unfortunately this is not the case for adding boundaries along the conditioning set variable axes for calculating the *conditional* posterior probability of independence. It is not difficult to construct artificial examples where that may be demonstrated. But the net effect is that increasing the number of conditioning values can increase or decrease the probability of independence, as judged by the resulting multidimensional histogram. Investigating the behavior with respect to conditioning is one of the future research directions we intend to follow.

An example run of our algorithm on two data sets, one independent by construction and one where X and Y exhibit a highly nonlinear dependency is shown in Fig. 4.3. In Fig. 4.3, bottom, we plot the evolution of $1 - p(t)$ of Fig. 4.2 as the XY plane is discretized at successively finer resolutions. Both plots demonstrate how the posterior probability of independence tends to unity as the grid becomes more complex. The plot on the right is not strictly "U-shaped" due to artifacts stemming from the finiteness of our data set. The top plots show the grid that produces the discretization of maximum dependence, *i.e.* $\Pr(M_I \mid \mathcal{D}) = 1 - p_{max}$. Note that the set of independent points contains only four bins. Our algorithm returns posterior probability of independence $1 - p_{max} = 0.88$ for the independent data set and 0.29 for the dependent one.

The algorithm of Fig. 4.2 essentially computes an approximation of the complete Bayesian integral over all possible grid boundaries at the resolution corresponding to the minimum probability of independence supported by our data set. At this limiting resolution $M^* \times M^*$ (assuming a square grid for simplicity), we treat the grid boundaries in a Bayesian fashion: ideally we should go over all possible $M^* \times M^*$ grids



(a) Data independent by construction

(b) Data nonlinearly dependent

Figure 4.3: An example run of our algorithm on two data sets. The actual data set, together with the grid that is output from our algorithm, drawn at the point of minimum independence. **Bottom plots:** The probability of independence as boundaries are added to the grid. The dashed horizontal line represents our prior $\wp = 0.5$.

whose boundaries lie on each possible pair of midpoints between successive data points along the axes. However, the number of such irregular $M^* \times M^*$ grids is very large, namely $\binom{N-1}{M^*-1}^2$, a number exponential in the number of data points. Our approach approximates this computation by maximizing over the first $M^* - 1$ boundaries and averaging over the M^*-th one. This results in a polynomial-time approximation of the Bayesian sum.

Our approach is expected to work in situations where the number of data points is large (greater than around 30) and the actual distribution does not look sufficiently close to an independent one at small numbers of data points (several hundred); if these assumption do not apply it may fail to recognize existing dependencies because the Bayesian treatment, all things being equal, contains greater weight to the simpler (independence) hypothesis. In Fig. 4.4 we see an example of such a failure: although X and Y are dependent in the form of a spiral, our procedure indicated independence, given 100 data points (probability of independence 0.77). We conjecture that the reason for this failure is that the spiral structure looks sufficiently similar to a set of concentric circles, approximating a circular Gaussian distribution (according to which, X and Y are independent). More work needs to be done in the future to identify and remedy the reasons for failures such

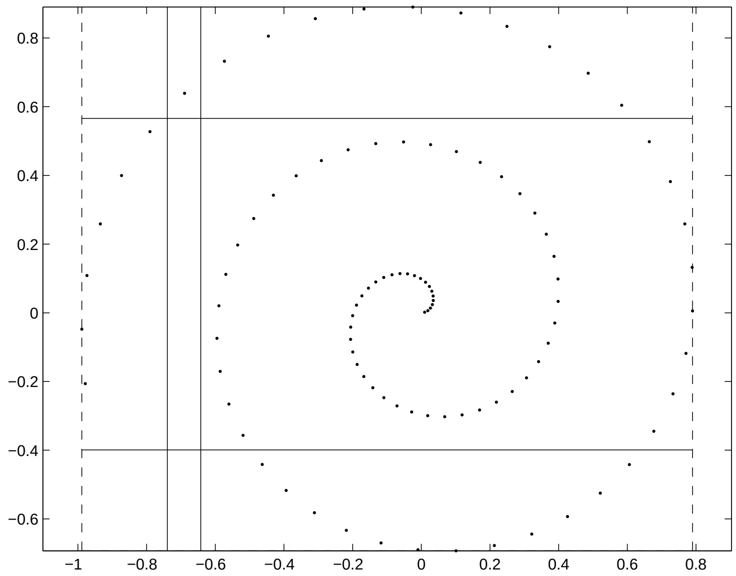


Figure 4.4: Probability of independence returned by the algorithm is 0.77, indicating independence (erroneously).

as these.

Our approach is non-parametric. It is well-known that parametric approaches are statistically less powerful (require more samples for the same level of discriminating power) than non-parametric ones. This is due to the fact that they make stronger assumptions on the form of the underlying model, and are thus able to represent it with fewer parameters, making it easier (in terms of number of samples) to estimate them. Our method is no different; if there exists prior knowledge of the form of the underlying joint distribution of X and Y (*e.g.* Gaussian, multinomial with a known number of bins *etc.*), it is always preferable to use a parametric test. Such a test would typically use the data set to calculate a set of confidence intervals for the parameters of that model, and deduce independence if the set of parameters that correspond to independence lie within these confidence intervals.

In summary, the justification of our algorithm is as follows: as we established by the structure of the resolution space, one needs to minimize posterior independence across resolutions, while approximating the average over an exponential sum over possible boundaries at each such resolution. Our algorithm accomplishes both goals simultaneously by maximizing dependence instead of minimizing independence across resolutions by employing an incremental maximum-value approximation to the evaluation of successive Bayesian integrals over multiple resolutions. In doing so, it reduces an exponential computation to a polynomial-order one.

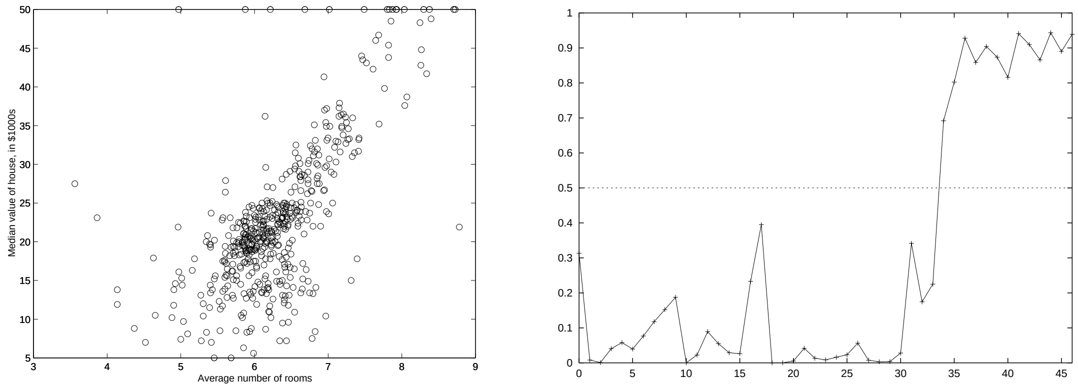


Figure 4.5: **Left:** Scatterplot of two dependent variables from the Boston housing data set, the average number of rooms per dwelling and and the median value of owner-occupied homes, in thousands of dollars. Spearman's rank correlation is very small (0.081) due to the fact that it is a two-dimensional distribution. **Right:** Plot of the evolution posterior independence probability as discretization boundaries are added during a run of our algorithm. The resulting posterior dependence probability is 0.99.

4.2 Experimental Results

In this section we compare our approach with existing ones in use routinely by statisticians. Perhaps the most frequently used method for non-parametric independence testing is rank correlation, either Spearman's rank order correlation coefficient r_s and Kendall's tau (τ) (Press et al., 2nd Ed., 1992). The two measures are related, and produce in essentially the same result in most cases. Here we will compare our test with Spearman's coefficient as it is the most expressive of the two. We will use some examples from the Boston housing data set, which is contained in the UC Irvine data set repository (Murphy and Aha, 1994).

In Fig. 4.5, we see two variables that appear dependent, the average number of rooms per house and its median value. The dependence between them is deemed very weak by rank correlation (its correlation coefficient is only 0.08), while our algorithm returns 0.99 as the posterior probability of dependence. We see the evolution of that probability for the first few steps on the right plot of Fig. 4.5. These variables were found dependent in the original study.

Some other examples of strongly dependent (Fig. 4.6, rank correlation is -0.02 while our algorithm returns 0.98) and slightly dependent pairs of variables (Fig. 4.7, rank correlation 0.01, posterior independence 0.41) demonstrate how our approach works as expected in practice, and better than existing established techniques.

4.3 Related Work

The most well-known independence test is perhaps the chi-square (χ^2) test. It operates on categorical data, and assumes there are enough counts in each bin such that the counts are approximately normally distributed.

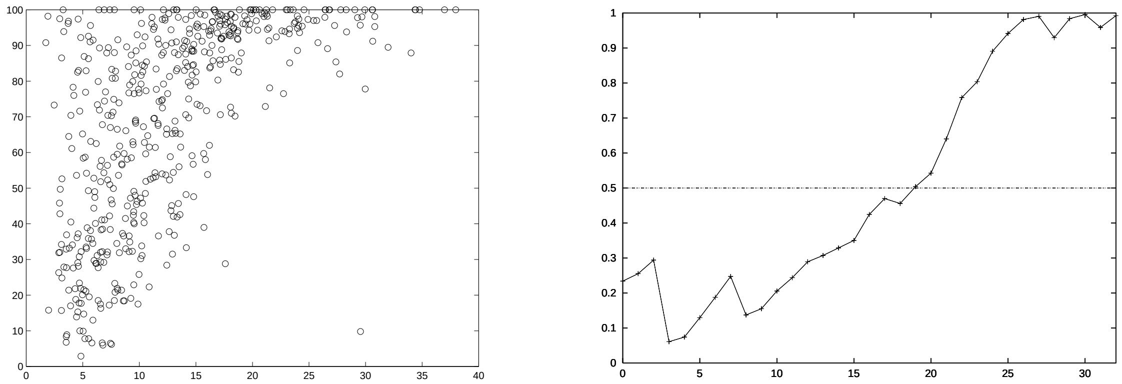


Figure 4.6: **Left:** Scatterplot of the percent lower status of population vs. proportion of houses built before 1940. Rank correlation indicates independence with a value of -0.02, while our method suggests dependence (0.77). **Right:** Posterior independence probability vs. number of discretization boundaries.

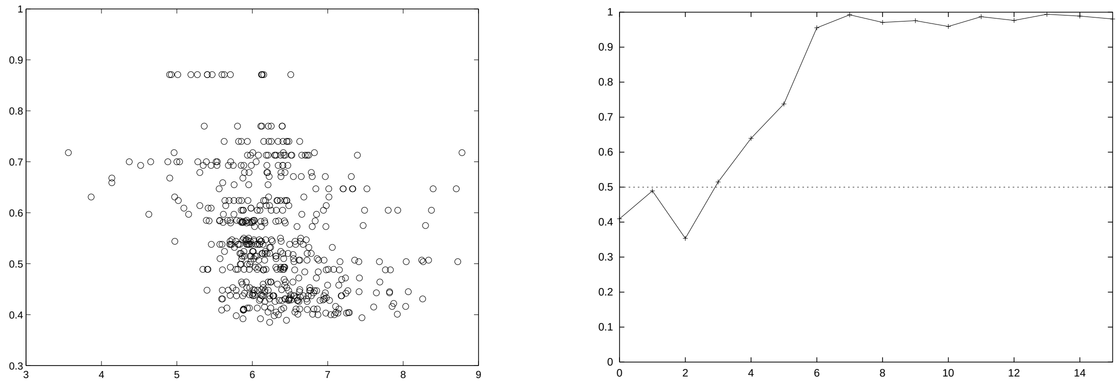


Figure 4.7: **Left:** On the X axis, the average number of rooms. On the Y axis, an air-pollution indicator. **Right:** The posterior probability of independence vs. the number of discretization boundaries. The variables are deemed independent by Spearman's rank correlation coefficient (0.01) but slightly dependent by our method (0.41).

Addressing the independence in 2×2 contingency tables, Fisher proposed an exact test in 1934 (see Agresti (1990) for one description). The difference between ours and Fisher's exact test is that the latter one is not Bayesian, and it computes a power value rather than the posterior probability. Bayesian approaches to contingency table independence also exist: the 2×2 case is addressed in Jeffreys' classic text (Jeffreys, 1961), and in general more recently (*e.g.* Gelman et al. (1995)), although not in exactly the same way as ours. However the greatest difference that makes our method unique is that it examines multiple resolutions and takes advantage of the ordering of the values of the continuous variables in the domain.

The most well-known statistical non-parametric tests of independence for ordinal variables are Spearman's rank correlation and Kendall's tau (Press et al., 2nd Ed., 1992). The former is the linear correlation of the relative ranks of the points along the X and Y axes, while the latter reduces ranks to binary or ternary variables (larger, equal, or smaller). Both tests can handle clear monotonic trends but fail to capture complex interactions between X and Y. Most notably, as demonstrated in Section 4.2, they also fail to capture joint probability functions of dependent variables that are two-dimensional *i.e.* distributions such that points sampled from them do not lie on or around a one-dimensional manifold.

Our approach is similar in many respects to Monti and Cooper (1998b). In that work, Monti and Cooper (1998b) present an iterative approach in which the learning of the structure of a Bayesian network in a domain with continuous and discrete variables is alternated with the enhancement (elaboration) of the discretization one of the continuous variables, during a combined search for the maximum score BN structure and discretization policy. Their score is the log-posterior probability of the data (in practice they use an appropriately adapted version of the BIC score). Our approach can be seen as a special case where the "Bayesian network" is a two-variable network containing X and Y only. The important difference to our approach is the fact that they iteratively attempt to elaborate the discretization of a continuous variable one at a time, while we add a boundary along each of the X and Y axes at the same time (and, in the case of a more general multi-variable dependence test, along the axes of each of the variables involved in the test). This might make a difference in cases where even though X and Y are dependent, there is considerable symmetry or repeating structure in the joint pdf to make the addition of a single variable boundary insignificant in terms of increasing the dependence of X and Y at the resulting discretization level. For example, a high-frequency sine-type functional dependence (a higher frequency version of Fig. 4.1) might prove difficult to discretize, especially for the beginning few boundaries, while adding two boundaries at a time might overcome this difficulty. More generally, this problem is identical to local maxima of the score in the approach of Monti and Cooper (1998b), where a two-step (or multiple-step in the case of a multiple-variable test) lookahead is needed to determine the existence of dependence and the most appropriate placement of the next set of boundaries. The problem of local maxima is noted in Monti and Cooper (1998b). Our approach sidesteps it by using an $O(n^2)$ search in the case of a two variable independence test. More research is needed in order to identify the class of distributions that are problematic with respect to iteratively discretizing one variable at a time.

As far as parametric approaches are concerned, there are a multitude of such approaches as they are the norm in practice. There are two main problems with all these approaches: first, they assume a certain family of distributions, which may fail in cases where the underlying data distribution does not fit well. Second, the question of proper bias and its trade-off with variance (the issue of model complexity) needs to be adequately addressed. This problem also occurs in our approach, and is solved naturally though the use of the Bayesian perspective.

With this chapter we conclude the theoretically-oriented part of the thesis. In the next chapter we focus on an important database application, presenting an effective solution involving structure learning of Bayesian network models.

Chapter 5

Structure Learning Application: Approximate DataCubes

5.1 Introduction and Motivation

In this chapter we will focus on the problem of estimating the result of count queries against a database approximately and fast. The problem of computing counts of records with desired characteristics from a database is a very common one in the area of decision support systems, On-Line Analytical Processing (OLAP), and data mining. A typical scenario is as follows: a customer analyst is interested in discovering groups of customers that exhibit an interesting or unusual behavior that might lead to possibly profitable insights into the company's customer behavior. In other words, a company wants to be able to *model* its customer base, and the better it is able to do that, the more insights it can obtain from the model and more profitable it has the opportunity to be. In this example scenario an analyst would, through an interactive query process, request count information from the database, possibly drilling-down in interesting subsets of the database of customer information. It is obvious that it is very important that the results to these queries be returned quickly, because that will greatly facilitate the process of discovery by the analyst. It is also important that they are accurate up to a reasonable degree, although it is not imperative that they are exact. The analyst wants an approximate figure of the result of the query and getting it correct down to the last digit is not necessary.

Our solution to the problem is motivated by these observations, *i.e.* that we need great speed coupled with only reasonable accuracy. In the following paragraphs we show that this is true for our method through performance results. In fact, our method can fit a database of billions of records in the main memory of a single workstation. This is due to the fact that we do not use the data to answer the query but only *a model of the data*. In doing so, our approach proposes a new viewpoint on the computation of DataCubes, one that advocates the use of models of the data rather than the data themselves for answering DataCube queries. Having said that however, the real challenge lies in how to construct a model of the data that is good enough for our purposes. For this, there are two important considerations that are relevant to the problem that we are addressing:

One, the model should be an accurate description of our database, or at the very least of the quantities derived from them that are of interest. In this problem these quantities are the counts of every interesting count query that can be applied to them (*i.e.* queries with some minimum support such as 1%; other query results can be due to noise and errors in the data). Second, the model should be simple enough so that using it instead of the actual data to answer a query should not take an exorbitant amount of time or consume an enormous amount of space, more so perhaps than using the actual database itself.

These two issues are conflicting, and the problem of balancing them is a central issue in the AI subfield of machine learning (which concerns itself with the development of models of data): it is always possible to describe the data (or the derived quantities we are interested in) better, or at least as well, with increasingly complex models. However, the cost of such models increases with complexity, in terms of both size to store the model parameters and time that it takes to use it for computing the relevant quantities (the query counts in our case). The reason we used Bayesian networks is their good performance in estimating pdfs in practice and their sound mathematical foundations in probability theory, as opposed to a multitude of other *ad hoc* approaches that exist in the literature. The method of producing the BNs from data that we use is a score-based algorithm (see Section 2.7.2).

5.2 Related Work

DataCubes were introduced in Gray et al. (1996). They may be used, in theory, to answer any query quickly (*e.g.* constant time for a table-lookup representation). In practice however they have proven exceedingly difficult to compute and store because of their inherently exponential nature (see example later in Fig. 5.1 and discussion in Section 5.3). To solve this problem, several approaches have been proposed. Harinarayan et al. (1996) suggest materializing only a subset of views and propose a principled way of selecting which ones to prefer. Their system computes the query from those views at run time. Cubes containing only cells of some minimum support are suggested by Beyer and Ramakrishnan (1999), who propose coarse-to-fine traversal that improves speed by condensing cells of less that the minimum support. Histogram-based approaches also exist (Ioannidis and Poosala, 1999), as well as approximations such as histogram compression using the DCT transform (Lee et al., 1999) or wavelets (Vitter and Wang, 1999). Perhaps closest to our approach is Barbará (1998), which uses linear regression to model DataCubes. Bitmaps are relatively recent method for efficiently computing counts from highly compressed bitmapped information about the properties of records in the database. They are exact techniques. Unlike the DataCube and Bayesian networks, bitmaps do not maintain counts, but instead perform a pass over several bitmaps at runtime in order to answer an aggregate query (Johnson, 1999; Chan and Ioannidis, 1999). Query optimizers for bitmaps also exist (Wu, 1999). Data mining research itself has been mostly focused on discovering association rules from data (Megiddo and Srikant, 1998; Bayardo Jr. and Agrawal, 1999), which can be viewed of as a special case of Bayesian network induction.

Generally speaking, Bayesian network research *per se* has flourished in the last decade, spurred mostly by Pearl's seminal book (Pearl, 2nd Ed., 1997). For more on BNs, see Chapter 2.

There has not been much work on using Bayesian networks for modeling databases. Notable exceptions are Getoor et al. (2001) which employs BNs to estimate the query selectivity (result size, in number of records), though they are concerned with using that result for query optimization and approximate query answering, and Silverstein et al. (1998), where possible causal relations from data are computed for purposes of data mining. Also, Davies and Moore (1999) used Bayesian networks for lossless data compression applied to relatively small datasets. Finally, Buntine (1991); Lam and Bacchus (1994b); Friedman and Goldszmidt (1997) present an approach for the related problem of online learning of the structure and parameters of a single Bayesian network model, by sequentially updating it when the data are abundant and for various reasons it is not desirable to use all of them at the same time. These reasons can range from the large size of the data set, making it impossible to be kept it in the main memory, to the problem that not all data may be available at the same time. The difference in the assumptions with our work is that instead we assume that all the data are available at the same time (batch learning).[1] Buntine (1991) maintain uses a continuous (never-ending) search in the space of BN structures, maintaining a subset of the sufficient statistics necessary for calculating the score of successor structures. Lam and Bacchus (1994b) on the other hand maintain a summary of past data in the form of a single BN constructed from the data seen so far. According to Friedman and Goldszmidt (1997), this approach does not work well because it is biasing learning toward the already learned structure. Mainly for this reason they propose an approach which stands in the middle ground between the approach of Buntine (1991) and Lam and Bacchus (1994b), maintaining both a BN structure summarizing past data and a set of statistics sufficient for calculating successors of that structure, invoking BN model selection each time a constant (user-specified) number of new data points is seen. They also mention that their approach can be extended by considering a different type of search (*e.g.* beam search) and thus maintaining a potentially smaller set of sufficient statistics. The relation of these approaches with ours is that in essence they are combining the data in a "linear tree" whereas we use a balanced, full tree combination. The advantage of using the latter approach is manifest in the case of a underlying distribution that is changing very slowly with time: in our method data from each time period will be used and their effect propagated to the root BN on an equal basis; for the "linear tree" approach that however is not the case since, as Friedman and Goldszmidt (1997) argues, independencies encoded in the structure of the BN

[1] Our approach can be conceivably adapted to a form of online learning by learning a new BN every time a number of new records arrive (*e.g.* at the end of the day in a retailer application) and invoking a recursive combination of the old BNs together with the new BN at that time. This is a topic of potential research.

that is representing previous data may possibly ignoring very weak dependencies that may exist and thus strongly bias learning toward preserving these independencies. Also, for the latter kind of approaches, there is a need to define a new and possibly *ad hoc* score of the combination of the existing BN (summarizing previous data) and the new data, which is a difficult problem and posed in Friedman and Goldszmidt (1997) as an open problem.

5.3 Bayesian Networks in Relation to DataCubes

In this section we emphasize the aspects of Bayesian networks that relate to our current application in decision support systems and data mining. We also discuss methods to automatically compute their structure from data samples (database records) taken from a domain of interest and we propose a new algorithm to attack the problem in the context of very large databases that cannot fit in main memory.

The attributes in the database are also referred to as "variables" throughout the chapter, since they have a one-to-one correspondence to the variables that appear in the Bayesian network.

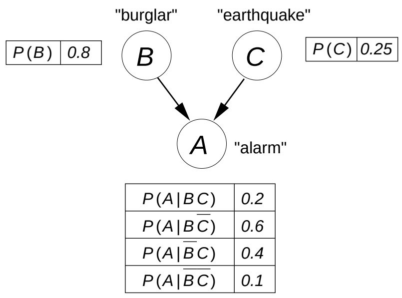
We illustrate the relation of DataCubes to BNs in a simple Bayesian network in Fig. 5.1, perhaps taken from the research department of a company that manufactures burglar alarms. It depicts three boolean variables, A ("home alarm goes off"), B ("burglar enters the house") and C ("earthquake occurs"). In this chapter we will assume that all variables are binary, although this is not necessary and does not affect the generality of our approach. In the binary case, each conditional probability table records the probabilities of that variable taking the value "true" for each possible combination of values ("true" or "false") of its parents. The meaning of the BN in Fig. 5.1 is that A depends on B and A depends on C but B and C are independent.

In general, a probability distribution can be specified with a set of numbers whose size is exponential in $|\mathcal{U}|$, namely the entries in the joint probability distribution table. One can represent such a table by a completely connected BN in general, without any great benefit. However, when independencies exist in the domain, using a BN instead of the full joint probability table results in two major benefits:

1. **Storage savings**. These may be significant to the point where infeasibly large domains may be representable, provided that they exhibit a sufficient number of independencies among the variables of the domain. The savings are typically exponential because conditional independencies are common in practice.

2. **Clear and intuitive representation of independencies**. Given the graphical representation of a BN, it is easy to determine the variables on which a quantity of interest depends on statistically and which are irrelevant and under what conditions.

Edge omissions indicate the existence of conditional independencies among variables in the domain. As mentioned above, if all variables in the domain statistically depend on all others, then there is no storage



(a) Bayesian network.

$\varnothing$	1000

A	435
$\overline{A}$	565

B	800
$\overline{B}$	200

C	250
$\overline{C}$	750

AB	400
$A\overline{B}$	35
$\overline{A}B$	400
$\overline{AB}$	165

$\overline{A}C$	60
$A\overline{C}$	375
$\overline{A}C$	190
$\overline{AC}$	375

BC	200
$B\overline{C}$	600
$\overline{B}C$	50
$\overline{BC}$	150

ABC	40
$AB\overline{C}$	360
$A\overline{B}C$	20
$A\overline{BC}$	15
$\overline{A}BC$	160
$\overline{A}B\overline{C}$	240
$\overline{AB}C$	30
$\overline{ABC}$	135

(b) Corresponding DataCube counts.

Figure 5.1: (a) Example Bayesian network and (b) DataCube constructed from a database of 1,000 examples. The Bayesian network consumes less space in this example because B and C are independent.

advantage to using a BN, since the storage required for the specification of the network is exponential in the number of attributes. Fortunately, in practice this is not the norm, and in fact the most interesting domains for data mining are those that exhibit a considerable number of independencies.

The storage space savings in this domain are illustrated in Fig. 5.1(b). There we see the complete DataCube of the domain using a database that contains 1,000 examples. The numbers that have to be stored in the DataCube are 22 essential counters. The numbers necessary in the corresponding BN are 6 probability entries. We see that for this particular example this is certainly not a significant improvement, especially considering the overhead of specifying the parents of each node and using floating point numbers for the probability entries. However, for large networks with tens or hundreds of variables, the savings increases exponentially if the corresponding network is sparse. For n attributes, the DataCube has to store 2^n tables of counts, with each table having size equal to the product of the cardinalities of the attributes they include (minus one). No full joint table for hundreds of variables containing either probabilities or counts could ever be stored using today's technology. However, such a domain can be succinctly represented by its joint probability distribution by taking into account the independencies that exist and using its Bayesian network instead. Such is the approach that we propose here.

The independencies expressed by a Bayesian network can be easily read from its structure using the rules of d-separation (see section 2.3). In Fig. 5.1 for example, B ("burglar") and C ("earthquake") are independent in the absence of any knowledge about A ("alarm"). If they were not, then either edge $B \to C$ or $C \to B$ would have to have been included in the network.

In general, conditional independence information can be very useful in practice in decision support systems and data mining applications. In the market basket paradigm for example, a customer that buys mouthwash may have increased probability of also buying shaving foam. However, the knowledge of the customer's age may make the probability of shaving foam purchase very unlikely (*e.g.* for very young customers). In this case, the local network structure that involves the three binary variables *Mouthwash*, *ShavingFoam* and *Age* would be $Mouthwash \leftarrow Age \rightarrow ShavingFoam$. According to this structure, *Mouthwash* and *ShavingFoam* are probabilistically dependent in the absence of any information, but are independent given information on the customer's *Age*.

5.4 Proposed Method

5.4.1 Problem Description

The problem we are addressing is the following:

> **Problem**: We are given a database that does not fit in memory and a procedure $BuildFromMemoryUsingData(\mathcal{D})$ that is able to generate a BN from a memory-resident database.
>
> **Desired Solution**: A representation that can fit in memory and a procedure $EstimateCount()$ that uses it to compute the approximate answer to count queries that may specify an arbitrary number of attribute values.

The naive DataCube solution is to preprocess and store the counts for all possible such queries (see example in Fig. 5.1). However, this is infeasible for almost any realistic domain. Compressed bitmaps are one way of answering such queries exactly. However they may exceed the main memory size for very large databases. Also, since their perfect accuracy is not needed in the kind of applications we are addressing, it is reasonable to trade-off a small amount of accuracy in exchange for a much smaller representation that can fit in main memory, which in turn translates to a significant benefit in query performance. Sampling is one approximate technique that is however linear time ($O(N)$) in the database size, as are bitmaps.

We propose to represent the record counts in the database with a single Bayesian network created from the entire database $\mathcal{D}$. Our method is *constant time* ($O(1)$) in the size of the database. It consists of merging a number of BNs, $\mathcal{B}_i$, each constructed from a partition $\mathcal{D}_i$ of the entire database into a single one, $\mathcal{B}$. Each $\mathcal{B}_i$ is created directly from $\mathcal{D}_i$ if it fits into main memory, or else by recursively splitting it, creating a network from each piece, and combining them in the same fashion that we combine the $\mathcal{B}_i$'s into $\mathcal{B}$. Each network $\mathcal{B}_i$ represents the joint probability distribution of partition $\mathcal{D}_i$. Since the $\mathcal{B}_i$'s are typically far smaller than the corresponding database partition $\mathcal{D}_i$, they can have the benefit of (1) simultaneously fitting into main memory and (2) tremendously speeding up the generation of the single network $\mathcal{B}$ since no disk access is required—we do not access the database during merging or at query time.

5.4.2 Notation

Before we present the algorithms, some notation: the entire database as a set of records is $\mathcal{D}$, and we denote each partition that we use to construct a Bayesian network $\mathcal{B}_i$ from as $\mathcal{D}_i$. Therefore $\bigcup_{i=1}^{N} \mathcal{D}_i = \mathcal{D}$ and $\mathcal{D}_i \bigcap \mathcal{D}_j = \emptyset$, for $i \neq j$. We want each partition $\mathcal{D}_i$ to be large enough so as to be representative, but small enough so that it fits into the main memory and satisfies the time constraints for building the corresponding Bayesian network. We have not done any studies on the size of each partition yet. In the experimental section 5.5 we used 100,000 records per partition.

In the next two sections we briefly discuss the *BuildFromMemoryUsingData*$(\mathcal{D}_i)$ procedure that produces a BN $\mathcal{B}_i$ from a memory-resident database $\mathcal{D}_i$, the *BuildFromDisk*$()$ procedure that merges them into a single BN, and the *EstimateCount*$()$ procedure that computes the count that corresponds to a user query.

5.4.3 Bayesian Network Structure from a Memory-Resident Database

The discussion of the previous section serves to establish the usefulness of modeling data domains using BNs. However, BNs are not as widely used as more traditional methods such as bitmaps for example, especially within the database community. We think that the main reason for this, apart from the fact that the database and machine learning communities are mostly separate, lies in the computational difficulty in inducing models from data. Bayesian network induction from data is not an exception. However, we argue that the benefits are great, especially in domains such as decision support systems and data mining where they are a natural fit. In this chapter we present an algorithm for computing and querying BNs constructed from very large databases that cannot fit in main memory, solving one of the main obstacles in adopting such a promising approach to many important problems in the database community.

The algorithm *BIChillclimb*$()$ described in Section 2.7.2 and shown in Fig. 2.4 is the one we used in our approach. It is implemented within the *BuildFromMemoryUsingData*$(\mathcal{D})$. Within this procedure, the

ProbabilityTables() procedure is a straightforward maximum-likelihood estimation of the probability entries from the database, which consists of counting the number of database records that fall into each table entry of each probability table in the BN. The score that is used and the probability table computation are central points in our approach since they are the only places in any of our algorithms (preprocessing or querying) that the database $\mathcal{D}$ is accessed. We use the BIC score:

$$ScoreFromData(B,\mathcal{D}) = BICscore(B,\mathcal{D}) = -\sum_{i=1}^{N} p_i \log p_i - penalty(B,N)$$

$$\begin{aligned} p_i &= \Pr((x_1,x_2,\ldots,x_n)_i \mid B) \\ &= \prod_{j=1}^{q_i} \Pr(X_j = (x_j)_i \mid \mathbf{Pa}_j, \mathcal{D}) \end{aligned}$$

$$penalty(B,N) = (|\mathcal{T}|/2)\log N$$

where $\mathbf{Pa}_j$ is the set of parents of variable X_j in B, the conditional probability $\Pr(X_j = (x_j)_i \mid \mathbf{Pa}_j, \mathcal{D})$ is computed by simple counting within the database $\mathcal{D}$, and $|\mathcal{T}|$ is the number of necessary entries in all tables of B (the number of essential parameters of the model B). We see that the score has two components: one that describes how well the network B describes the data ($-\sum p_i \log p_i$) and one that penalizes B for being too large (see discussion in Section 5.1 for details).

In Section 5.4.4 we will describe how to compute the BIC score, and in particular the first term, from a set of Bayesian networks that represent our database, instead of the records themselves. This is necessary when merging BNs representing portions of $\mathcal{D}$ into a single BN, without accessing the data. In Section 5.4.4 we also show how to implement the *ProbabilityTables*() procedure without accessing the database.

5.4.4 Algorithm for preprocessing the database: building and merging the BNs

The proposed *BuildFromDisk*() procedure to build Bayesian network $\mathcal{B}$from data stored on disk is shown in Fig. 5.2. Inside the procedure, the *BuildFromMemoryUsingData*() procedure contains the implementation of the algorithm for finding the structure of a Bayesian network from data that was described in Section 5.4.3. We note that the generation of each $\mathcal{B}_i$ can be done in parallel.

Having produced the networks $\mathcal{B}_i, i = 1,\ldots,K$, we combine them into a single one, $\mathcal{B}$, using the following procedure:

Procedure $(\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K) = \mathit{BuildFromDisk}(\mathcal{D})$:

1. Partition the database $\mathcal{D}$ into N equal partitions $\mathcal{D}_i, i = 1, \ldots, K$ so that each fits in main memory. Let $d = |\mathcal{D}_i|$, for all i.
2. For each $i = 1, \ldots, K$ do the following:
   (a) Read $\mathcal{D}_i$ into memory.
   (b) Build Bayesian network $\mathcal{B}_i$ from $\mathcal{D}_i$: $\mathcal{B}_i = \mathit{BuildFromMemoryUsingData}(\mathcal{D}_i)$.
3. Merge the networks $\mathcal{B}_i$ into a single one: $\mathcal{B} = \mathit{RecursivelyMerge}(\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K)$.

Figure 5.2: Algorithm for preprocessing the database.

Procedure $\mathcal{B} = \mathit{RecursivelyMerge}(\mathcal{B}_1, \ldots, \mathcal{B}_K)$:

If $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K$ simultaneously fit in main memory then:
  $\mathcal{B} = \mathit{BuildFromMemoryUsingBNs}(\mathcal{B}_1, \ldots, \mathcal{B}_K)$
else:
  $\tilde{\mathcal{B}}_1 = \mathit{RecursivelyMerge}(\mathcal{B}_1, \ldots, \mathcal{B}_{\lfloor \frac{K}{2} \rfloor})$.
  $\tilde{\mathcal{B}}_2 = \mathit{RecursivelyMerge}(\mathcal{B}_{\lfloor \frac{K}{2} \rfloor + 1}, \ldots \mathcal{B}_K)$.
  $\mathcal{B} = \mathit{RecursivelyMerge}(\tilde{\mathcal{B}}_1, \tilde{\mathcal{B}}_2)$.

The $\mathit{BuildFromMemoryUsingBNs}(\mathcal{B}_1, \ldots, \mathcal{B}_K)$ procedure is the only remaining one that needs to be defined. It is exactly the same as the $\mathit{BuildFromMemoryUsingData}(\mathcal{D})$ one (see Section 5.4.3), with the exception that the score is now computed from the BNs ($\mathit{ScoreFromBNs}()$ procedure) that are its arguments instead of the database ($\mathit{ScoreFromData}()$ procedure):

$$
\begin{aligned}
\mathit{ScoreFromBNs}(\tilde{\mathcal{B}}, \overbrace{\mathcal{B}_1, \ldots, \mathcal{B}_K}^{\text{representing } \mathcal{D}}) &= \sum_{t \in \mathit{Tables}(\tilde{\mathcal{B}})} \log \Pr(t \mid \tilde{\mathcal{B}}) \\
&\quad \times \left[(1/K) \sum_{k=1}^{K} \mathit{EstimateProbability}(t, \mathcal{B}_k) \right] \\
&\quad - \mathit{penalty}(\tilde{\mathcal{B}}, N).
\end{aligned}
$$

In the above formula the outer sum goes over all table entries t in $\tilde{\mathcal{B}}$. Each such table entry corresponds to a configuration of variable assignments (for the node and the parents of the node that it is attached to) and "don't cares" (for the remaining variables in the domain)—see Fig. 5.1 for example. The inner equally-weighted sum is simply an average over all networks $\mathcal{B}_i, i = 1, \ldots, K$ of the probability of that configuration. $\Pr(t \mid \tilde{\mathcal{B}})$ is the probability of configuration t in $\tilde{\mathcal{B}}$, and can be read directly off the corresponding table entry of $\tilde{\mathcal{B}}$. The *EstimateProbability*() procedure is taken from the literature; choices are discussed below in Section 5.4.5.

The computation of the probability tables by the *ProbabilityTables*() procedure is also done from the $\mathcal{B}_i$'s without accessing the database; it is making use of the *EstimateProbability*() procedure:

$$\forall t \in Tables(\tilde{\mathcal{B}}), \quad \Pr(t) = (1/K) \sum_{k=1}^{K} EstimateProbability(t, \mathcal{B}_k)$$

Since the database access is $O(N)$ during the *BuildFromDisk*($\mathcal{D}$) procedure, the number of networks at the base of the recursion is $K = N/d = O(N)$, and since accessing a BN does not depend on the database size, it is easy to make the following observation:

> **Observation**: the entire *BuildFromDisk*() algorithm is $O(N)$ time (linear in the size of the original database) and thus scalable. Moreover, it is parallelizable, with a straightforward parallel implementation.

This observation is supported by the experimental results (Section 5.5, Fig. 5.8).

5.4.5 Algorithm for answering a count query from a Bayesian network

After generating a single BN for our database, we can use it to answer count queries. In order to do that, we need to estimate the probability (expected frequency) of the query using the BN, and multiply it with the number of records in the database (see Section 5.4.5). *We do not need to access the database* for this.

The computation of this probability may be involved and in general cannot be simply read off the probabilities in the tables of the network. For example consider two variables X and Y that are very far apart but connected by a directed path. The probability of $X = 0$ and $Y = 1$ without knowledge of the value of any other variable in the network is not a simple function of the entries in the conditional probability tables of the BN. Rather, it requires a process called probabilistic inference.[2]

There exist several algorithms for inference. Two kinds of methods exist: approximate and exact. Approximate ones (Henrion, 1988; Fung and Chang, 1989; Schachter and Peot, 1989) are sample-based, and generate an artificial database of samples during the process of estimation (the generated samples are discarded immediately and only the count of those than matched the query is kept). Their main disadvantage

[2] Which is a generalization of logical inference—given a BN, it computes the probability of the truth of a compound predicate (query) rather than a true/false value.

is that they are slow and may need a great number of samples to estimate the probability of the query to a sufficient degree. For exact inference, the most popular method is the join-tree algorithm. The details of the algorithm are beyond the scope of this work, see Pearl (2nd Ed., 1997); Huang and Darwiche (1994). Its running time depends on the number of variables and the complexity of the BN, but in practice for typical BNs of a few tens of variables it runs in under a second. This is the method we use here, contained in the $EstimateProbability()$ procedure.

In our approach, to estimate approximate counts for query Q from the Bayesian network $\mathcal{B}$that is is the output of the $BuildFromDisk()$ procedure, we use the $EstimateCount()$ procedure, shown below:

Procedure $\hat{N} = EstimateCount(Q, \mathcal{B})$:
$\hat{N} = N \times EstimateProbability(Q, \mathcal{B})$.

For the procedure $EstimateProbability()$ in our implementation we use the join-tree algorithm. $EstimateProbability()$ returns the probability of query Q according to the probability distribution represented by $\mathcal{B}$. Since $\mathcal{B}$ is a representative of the N records contained in $\mathcal{D}$, $N \times EstimateProbability(Q, \mathcal{B})$ is an estimate of the number of records within $\mathcal{D}$ for which Q evaluates to "true."

Since the $EstimateCount(Q, \mathcal{B})$ algorithm does not access the database, under our assumptions we can make the following observation.

Observation: the $EstimateCount(Q, \mathcal{B})$ procedure is $O(1)$ time in the size of the database.

5.5 Experimental Results

We experimentally tested our approach on real and synthetic data. The real data consists of customer information data, obtained from a large anonymous retailer.[3] It consists of over 3 million customer transactions (3,261,809) containing information on whether the customer purchased any of the 20 most popular items in the store. The data represents one week of activity and its concise representation occupies around 8 MB. This size coincides with the size of its uncompressed bitmap. Although this database is not large in size, we use it in order to obtain performance results on the compression ratio we can hope to obtain on real-world data.

In order to assess the scalability of our system, we needed larger sets that were not available at the time of our evaluation. For this reason we used synthetic data for our scalability study. All remaining experiments,

[3] For confidentiality reasons we cannot reveal the name the retailer nor the products involved.

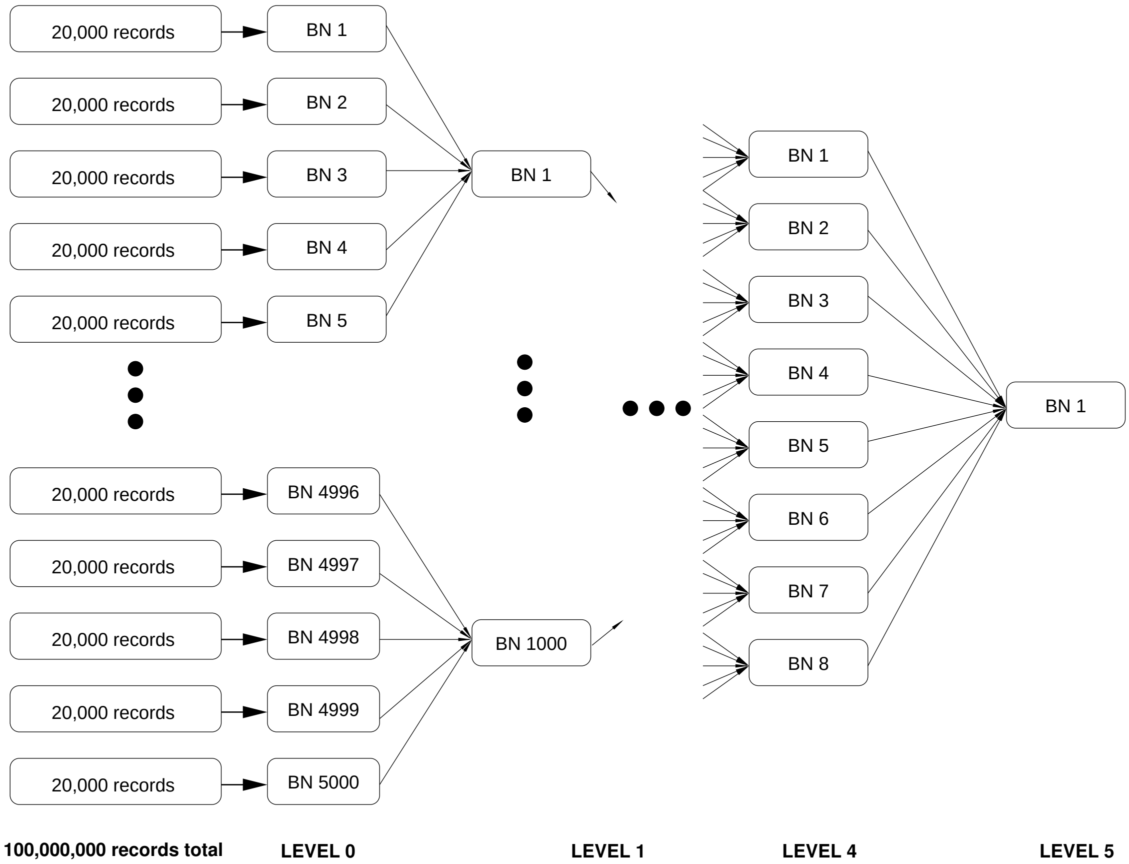


Figure 5.3: Illustration of the recursive combination of the QUEST database at 6 levels. At every level five networks are combined, with the exception of the last level where eight networks were combined.

besides the compression size results, used the synthetic data. These were produced by a program available from IBM's QUEST site.[4] The generation program produces a large, user-specified number of random association rules involving a number of attributes (their number is also randomly distributed around a user-specified mean), and then generates market-basket data whose statistical behavior conforms to those rules. We produced a database of 100 thousand and 1, 10, and 100 million records from a store inventory of 5,000 items (products) using 10,000 customer patterns having an average length of 4 items. (Each customer pattern corresponds to an "association rule.") The average transaction length was 10 items. Contrary to our real database, we used the 50 most frequently used items. Such a DataCube cannot fit in main memory since it consists of 2^{50} entries.

From both real and synthetic databases we then constructed a number of Bayesian networks from that data in order to model their joint probability distribution. We split the data set in a number of subsets $\mathcal{D}_i$, each containing at most d records, where $d = 815,452$ for the anonymous retailer data set (4 chunks) and $d = 20,000$ for the QUEST data set (5,000 chunks). We then used each subset $\mathcal{D}_i$ to construct the corresponding Bayesian network $\mathcal{B}_i$. Finally, we recursively combined the networks using a two-level hierarchy for the real data set and a six-level hierarchy, depicted in Fig. 5.3, for the QUEST data set. For the latter, at every level five networks are combined, with the exception of the last level where eight networks were combined.

We compare our results against an uncompressed and compressed bitmap, as well as compressed a bitmap produced after sampling the database for 1% and 10% of its records uniformly.

Our experiments evaluate our approach with respect to the following dimensions:

1. Query count error.
2. Space to store models and effective compression of the database.
3. Time to answer a query.
4. Build time and scalability.
5. Visualization of the dependencies in the database.

Because the number of possible queries grows exponentially with the number of variables that are allowed to be involved in it, we were not able to perform all possible queries of any sizable length. Instead we generated 10,000 random queries of length up to 5 variables except in the case of averaging over networks produced at levels 0 and 1, where only 300 and 1,000 random queries were used, respectively, due to the excessive time needed to process each query at that level. Each query is more general than one traditionally used in association rule discovery, allowing testing for the presence *or absence* of any particular item in a transaction, from the 20 or 50 most frequently purchased items. For example one such query may be "what

[4] `http://www.almaden.ibm.com/cs/quest/`

Database	Records	Bitmap size (bytes)	Compression ratios (*before*:*after*) gzip	bzip2	Sampled 10% & bzip2	NetCube	
QUEST	20,000	125,009	4.2:1	4.4:1	40:1	**85:1**	(**1,469 bytes**)
	100,000	625,010	4.3:1	4.5:1	42:1	**523:1**	(**1,195 bytes**)
	500,000	3,125,010	4.4:1	4.5:1	43:1	**2,741:1**	(**1,140 bytes**)
	2,500,000	15,625,011	4.4:1	4.5:1	43:1	**7,508:1**	(**2,081 bytes**)
	12,500,000	78,125,012	4.0:1	4.5:1	44:1	**26,050:1**	(**2,999 bytes**)
	100,000,000	625,000,013	4.4:1	4.5:1	45:1	**1,211,477:1**	(**5,145 bytes**)
Anonymous retailer	3,261,809	8,154,540	3.8:1	3.8:1	37:1	**1889:1**	(**4,317 bytes**)

Table 5.1: Comparison of compression ratios for various databases used for the experiments. The first rows correspond to the QUEST-generated databases while the last one corresponds to real data obtained from an anonymous retailer. The sampling figures refer to 10% sampling and after `bzip2` compression. For the NetCube, the trend of compression ratios that are increasing with database size is due to increasing benefits from using an approximately fixed-sized probabilistic model of a domain in place of data drawn from it.

is the number of transactions in the database in which a customer purchased milk and orange juice but not bread?"

5.5.1 Compression

In this set of experiments we compare the size of our representation to that of compressed bitmaps and sampling by 10%, also compressed. Compressing the bitmaps of each of our databases produced approximate 7:1 compression ratio for the synthetic QUEST databases and 3.8:1 for the real-world data. Compressing the sampled database predictably produces linear compression with respect to compressed bitmaps. In contrast, the NetCube approach typically produced compression ratios of 85:1 to 1,211,477:1 for synthetic data and 1800:1 or more for real data. The compression ratios and BN sizes are shown in Table 5.1 and are also plotted in Fig. 5.4. The price for such a high compression performance is the fact that it is lossy. However, if the user can tolerate a certain amount of error (see below), then it may be the method of choice for the data analyst, since its space requirements are modest and has the inherent advantage of visualization.

Note that the network produced from real data, corresponding to one week of transactions, occupies only 4 KB. If are allowed to make the conservative assumption that the network from any given week is 10 times this one (40 KB), and the assumption that doubling the database size doubles the size of the resulting network (for which our experiments have no support of, and in fact indicate that it might not grow at that rate but a much smaller one), then our approach makes it possible to fit 20 billion transactions in the memory of a regular workstation with 256 MB of main memory, corresponding to more than 100 years of transactions at this rate, effectively spanning the lifetime of most businesses.

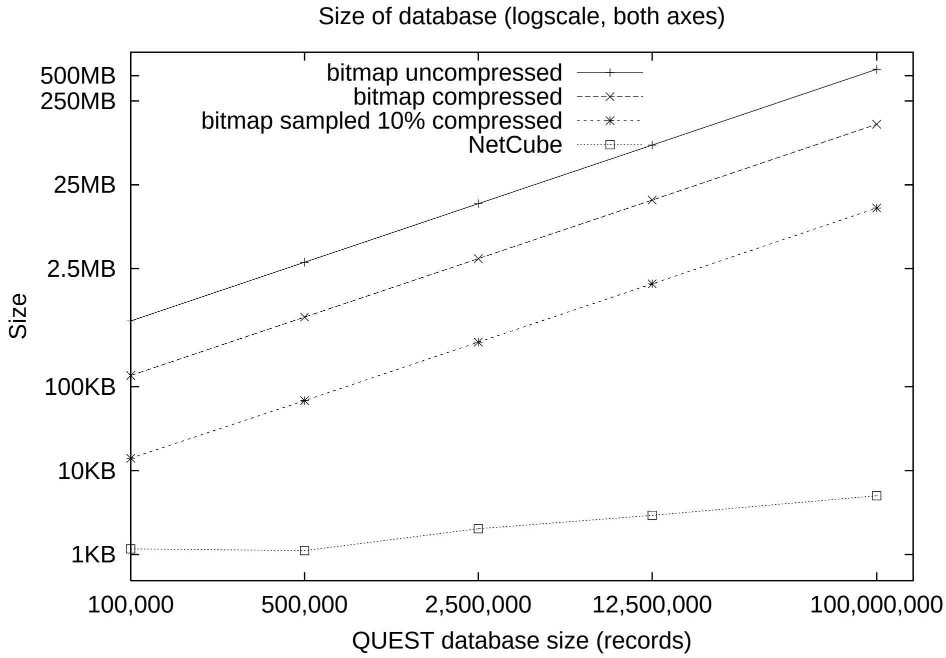


Figure 5.4: Comparison of the size of the compressed database using bitmaps, sampling by 10% and NetCubes. The difference between `gzip` and `bzip2` is small (see Table 5.1), so only the best of the two (`bzip2`) is used here.

5.5.2 Query time

We used a workstation with 128 MB of physical memory for our query time experiments. Running our set of queries on the bitmaps we noticed a slowdown for the larger QUEST databases whose bitmap cannot fit into main memory. This happens because the bitmap system had to use part of the virtual memory system which resides on the disk (thrashing). An important observation we can make here is that although bitmap compression will temporarily alleviate this problem, a database of more than 4.5 times our largest one would again force the bitmap method into the same thrashing behavior (note the compression ratio 4.5:1 for bitmaps in Table 5.1). A database of such a problematic size would not be atypical in today's real-world problems.

We plot query times in Fig. 5.5. As we can see in general query times are modest except in the case of the NetCube at levels 0 and 1, and compressed bitmaps. Most of the time for queries on bitmaps is used for loading the bitmap into memory (although thrashing is also a problem), and this can only become worse as the size of the database grows since the bitmap size grows linearly. The NetCube at levels 0 and 1 does poorly due to the large number of BN models it needs to query.

In Fig. 5.6 we show the performance of querying using the NetCube as a function of the level that we use

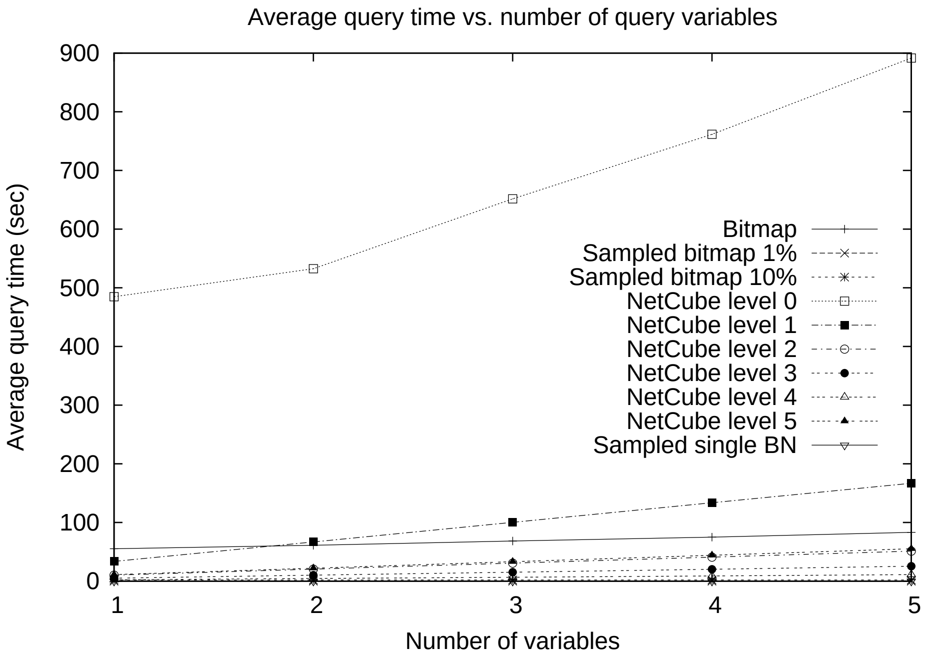


Figure 5.5: Average query times for the QUEST data set as a function of the number of variables in a query.

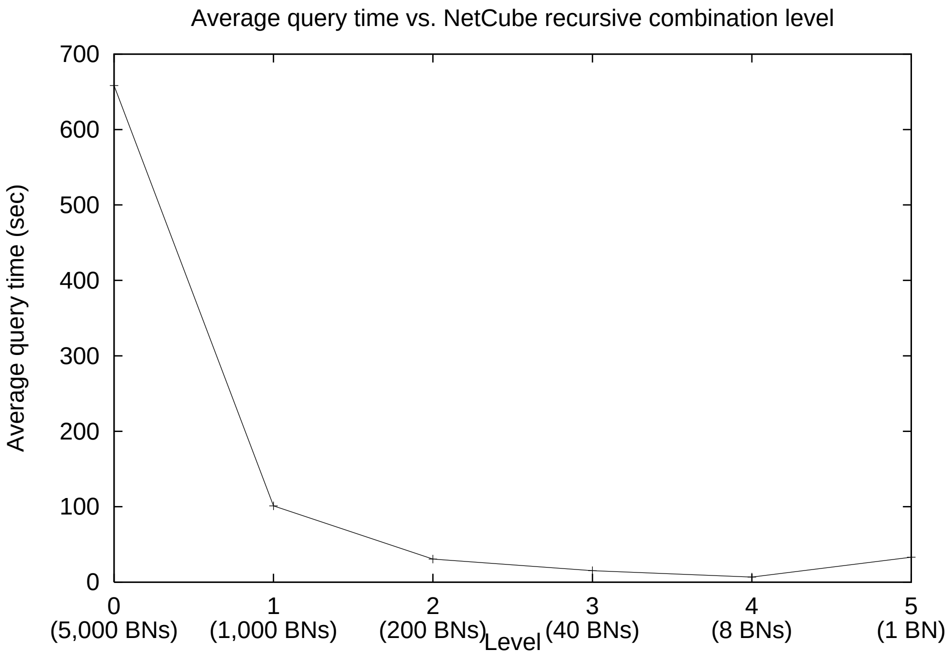


Figure 5.6: Average query times for the QUEST data set as a function of the recursion level that was used to answer queries.

to answer the query. For example, using only a level 1 NetCube for a QUEST query would use 1,000 BNs at that level each being a representative of 100,000 records in the database. As was noted above, using a low level incurs a large performance penalty due to the large number of BNs that we need to use inference on in order to answer the query. This is especially dramatic in the case of level 0, where 5,000 BNs need to be queried. On the other end of the spectrum, a single BN used in level 5 is not ideal either, because it is significantly more complex (densely connected) and thus a query takes more time. As Fig. 5.6 suggests, the ideal level to use in this case is 4, which represents an interesting balance between the number of BN models and their complexity.

5.5.3 Query error

We conducted an assessment of the query error using a set of 10,000 random queries containing up to 5 variables (only 300 queries were used for "NetCube level 0" and 1,000 queries for "NetCube level 1" due to large running times). Because relative error becomes artificially large for queries of very little support even when the count difference is not very large, we used queries that had at support of 10,000 records or more. Apart from artificially weighing the error rate, queries of very small support are arguably "uninteresting" and/or can be due to spurious factors. Such treatment is consistent with other approaches in the literature (*e.g.* Beyer and Ramakrishnan (1999)). Note that our minimum support is not a percent of the entire database; this means that our assessment applies to cases where the user is looking for subtle events even as the database size increases. We used 10,000 as the cutoff for the support of these subtle events but more research must be done to determine this threshold of significance.

To get an idea of the effect that the chunk size has on the query error performance, we compare against another approach, namely sampling 20,000 records from the QUEST database (the number of records used in level 0 of the recursive combination depicted in Fig. 5.3) and producing a Bayesian networks from those. To answer a query using this method we use only the resulting network as our estimate of the joint pdf. This is called "sampled single BN" in the results of Fig. 5.7.

In Table 5.2 we list the percentage of queries that achieve error 5% or less, for each method. That is to be expected given the levels of compression that are achievable. From the table we see that the NetCube does not achieve a good accuracy when using levels greater than 0. The actual error distribution is shown in detail in Fig. 5.7. In that we can verify that there is significant mass at error levels less than 50%. We also notice that the NetCube at level 0 as well as the sampled single BN exhibit an increase in errors at around 200%, meaning that they return double the actual count of records of the database. Both of these approaches essentially use 20,000 records per model (answering a query using a NetCube at level 0 corresponds to essentially averaging over all level 0 BN models, each constructed from 20,000 records). Using higher levels of the NetCube alleviates this particular behavior, and is consistent with the fact that they take into account a larger number of records when creating the model, perhaps discounting certain transient effects which might

Method	95-percentile of query errors (%)
Bitmap	100
Sampled bitmap 10%	95
Sampled bitmap 1%	88
NetCube level 0	70
NetCube level 1	33
NetCube level 2	33
NetCube level 3	33
NetCube level 4	33
NetCube level 5	32
Sampled single BN	56

Table 5.2: Percent of queries for which the accuracy is at most 5%.

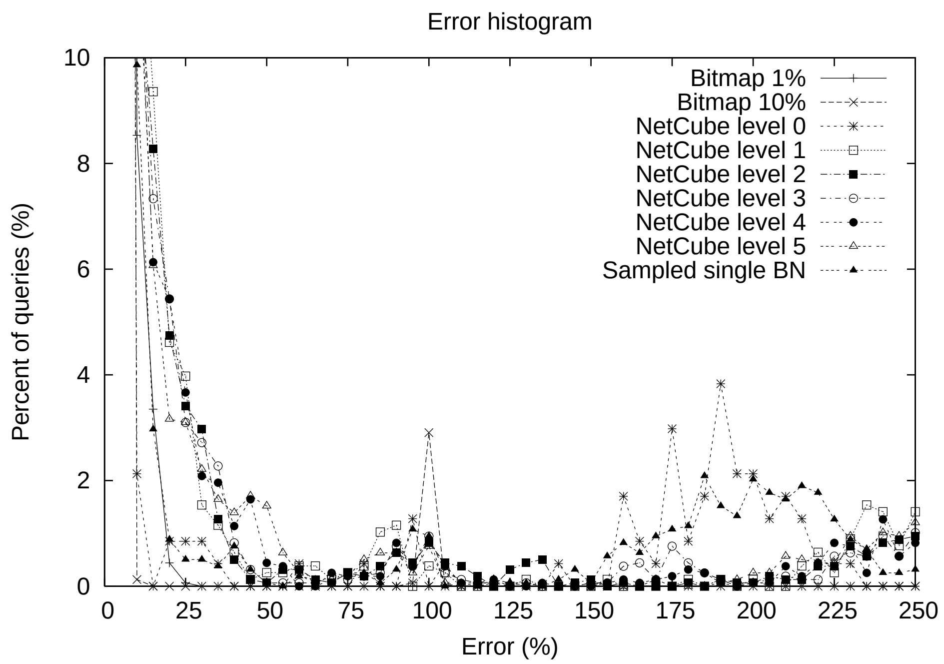


Figure 5.7: Error distribution for different approaches.

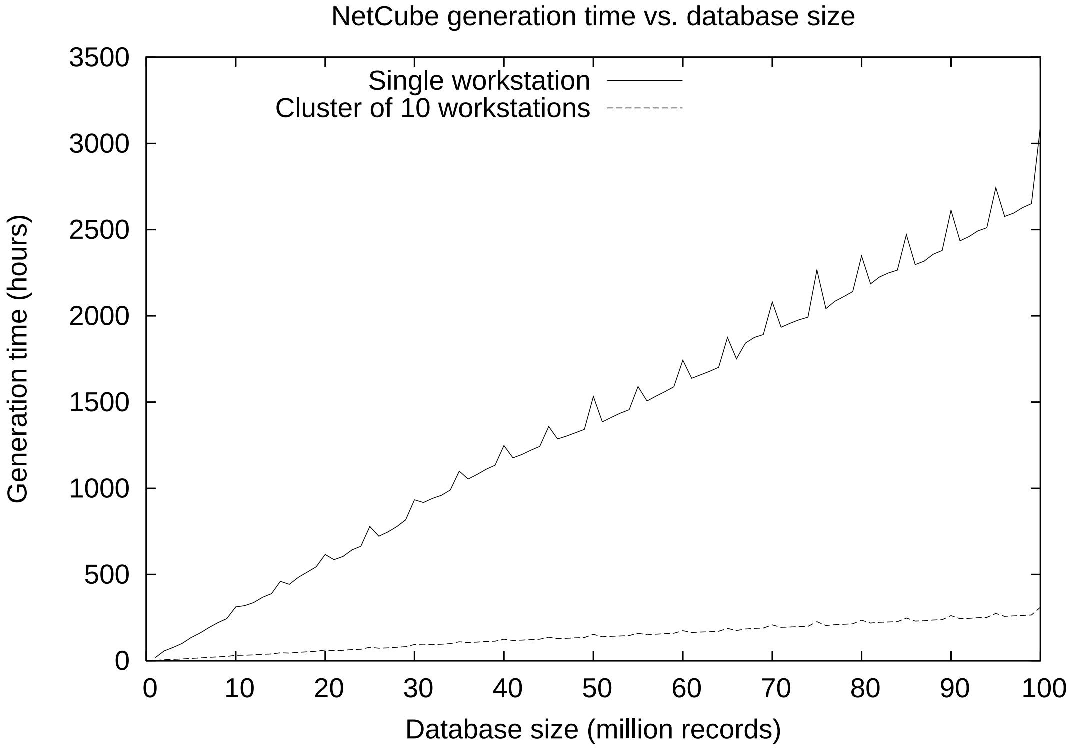


Figure 5.8: Build time for the set of multiple BNs increases approximately linearly with the number of records in the database. Parallelization over a number of workstation scales the build time down linearly.

be present when using 20,000 records only. Further investigation and a detailed analysis of phenomena of using different database chunk sizes and datasets that contain transient changes in the distribution or slow "drifting" of the distribution is the subject of future work.

5.5.4 Build time

As mentioned above, we generate a BN for each database piece of 20,000 records. As we can see in Fig. 5.8, our method is linear on the database size, and thus scalable. As can be seen in that figure, there exists a number of jumps at certain sizes; these correspond to additional time spent combining BNs from lower levels (*i.e.* levels closer to the data in Fig. 5.3) to the next higher one, and occur at 100,000, 200,000 *etc.* records (level 0 to level 1), 500,000, 1,000,000 *etc.* records (level 1 to level 2) *etc*. However since the total number of nodes in the recursion tree of Fig. 5.3 is at most twice the number of nodes at level 0, the overall build time is also linear in the database size.

Each database piece can be processed in parallel, and the merging of the BNs can also be done in parallel across the same recursion depth. Thus our method is parallelizable in a straightforward manner. Parallelization over a cluster of workstations scales linearly, making the generation of a database of 200 million transactions a matter of hours on a modest cluster of 10 workstations, as shown in Fig. 5.8.

We note here that our attempts to create a single BN by using the straightforward *BuildFromMemoryUsingData*() algorithm on the entire database were unsuccessful for very large problems of size 100 million records or more; the algorithm did not terminate while producing the network after 4 days and had to be manually aborted. This underscores the usefulness of using our recursive combination procedure (*BuildFromDisk*() procedure) for any kind of practical application that involves very large databases.

5.5.5 Visualization

In Fig. 5.9 we show a BN produced from real data corresponding to a week of activity of the 20 most frequently purchased items at a large anonymous retailer. We also show the network produced at level 5 (top level) using the 50 most frequently used attributes of the QUEST data set. The advantage of the graphical representation of the BN that our approach generates is that it can be used to clearly depict variables that are the most influential to the ones that the analyst might be examining. Moreover, the conditional probability tables will give our analyst the exact nature and strength of these influences. Therefore our approach fits well in the data mining procedure and can save the analyst large amounts of time that would be otherwise spent on exploration, drill-down analysis *etc.* of the customer database.

5.6 Summary

In summary, the characteristics of the method are:

Small space: the resulting BN takes up a tiny fraction of the space that the original data that are queried upon. We produced greater than 1800:1 compression ratios on real data.

Scalability: we can handle databases of arbitrarily large number of records; the method's preprocessing time scales linearly with the size of the database. Moreover, it is parallelizable with a straightforward parallel implementation.

Query time: the method can answer arbitrary queries in a short time when used appropriately *i.e.* a few seconds for a NetCube of level 4.

Accuracy: the method has reasonable accuracy when using low levels (closer to the data) to answer queries. More research is needed for effective error reduction when higher level, recursively combined BN models are used.

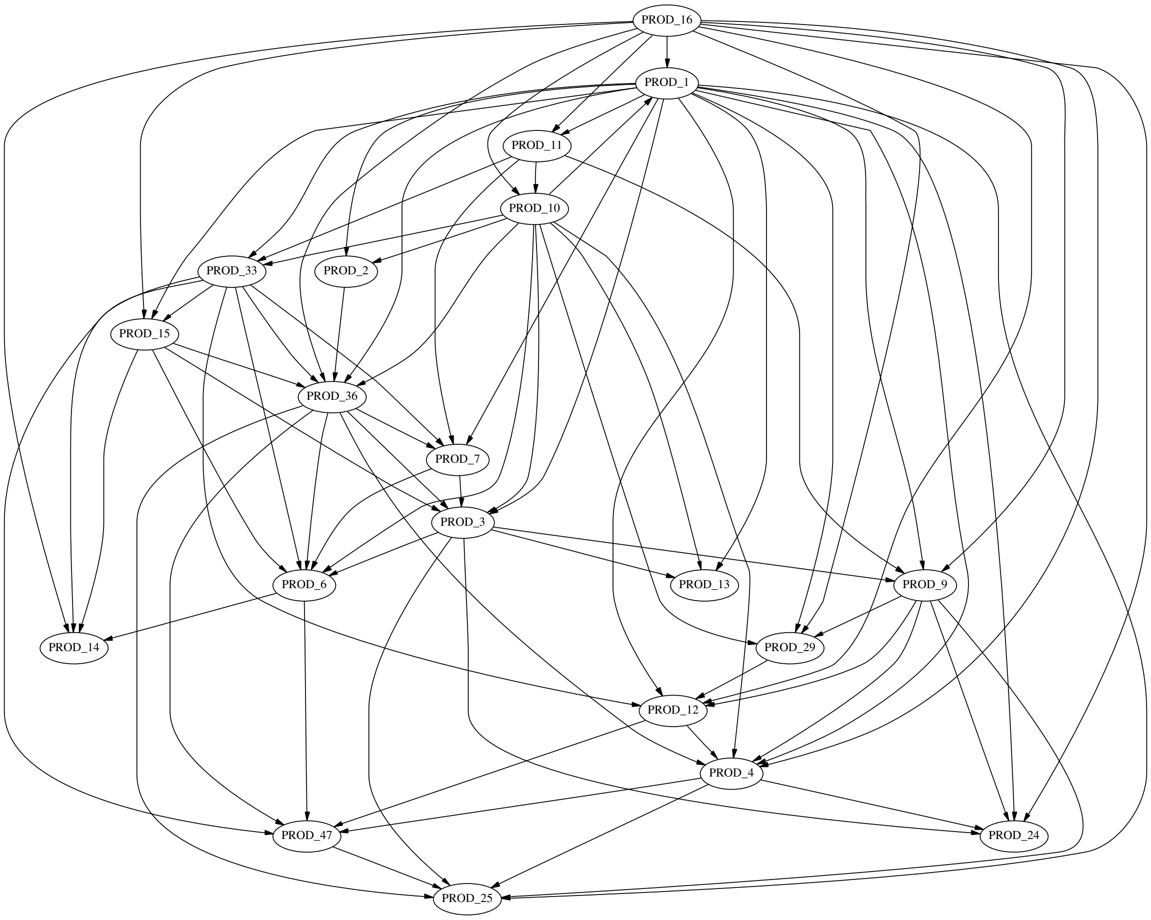


Figure 5.9: Bayesian network produced from real data obtained from a large anonymous retailer. For confidentiality reasons, we have anonymized the names of the products that are displayed in the graph.

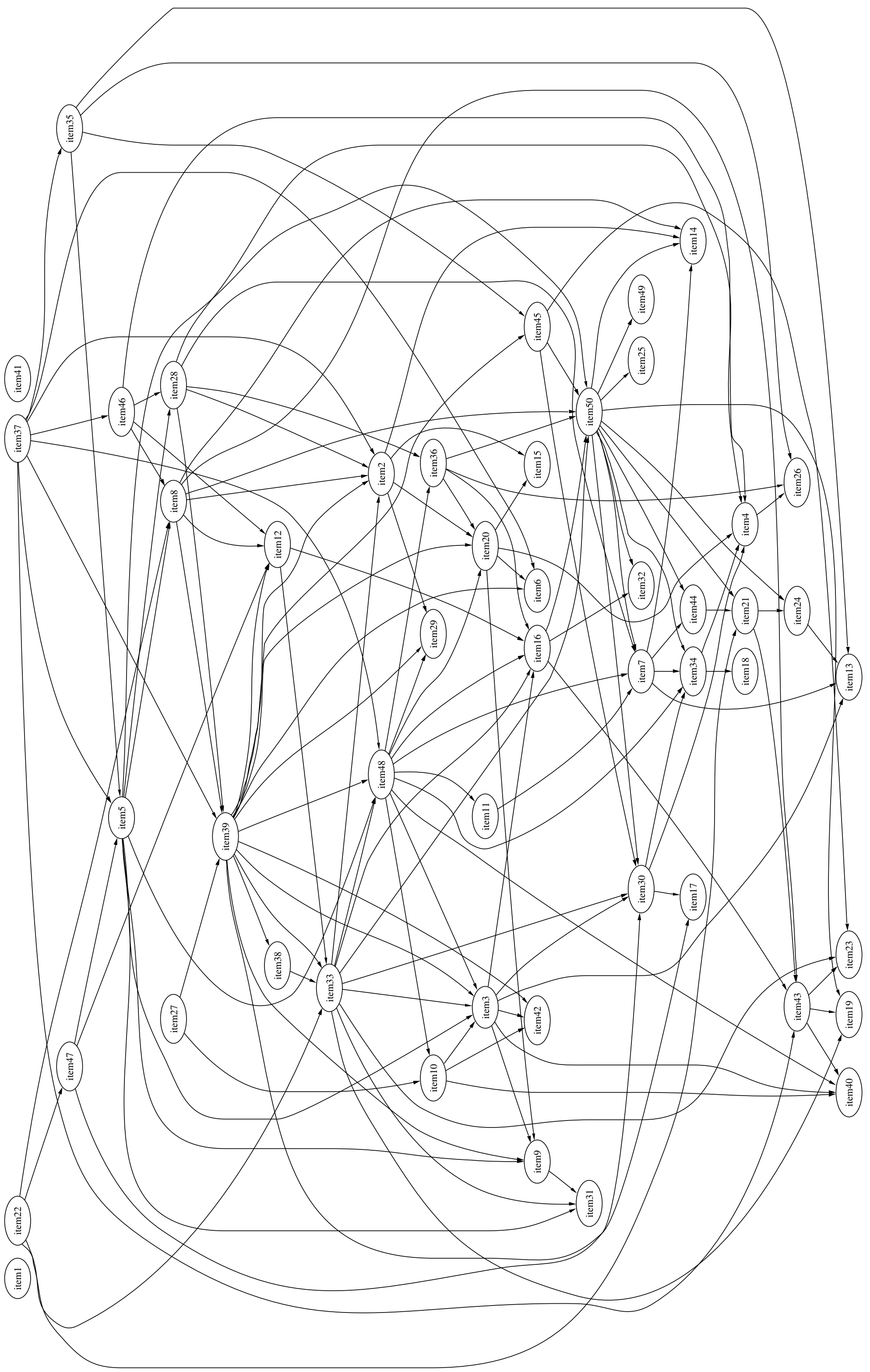


Figure 5.10: The final Bayesian network produced at level 5 from the QUEST data set.

Suitability to data mining: the representation that is used by the algorithm, namely Bayesian networks, are an excellent method for visually eliciting the most relevant causes of a quantity of interest and are a natural method to support data mining.

In the final chapter we present the conclusion of the thesis and avenues of future research.

Chapter 6

Conclusions and Future Research

In this thesis we focused on approaches to solving three distinct tasks, all of which involved learning the structure of Bayesian networks from data.

First, we presented an independence-based approach to learning the structure of Bayesian networks efficiently. We first developed an efficient algorithm for computing the Markov blanket of a node. We then proposed the GS algorithm, which uses the Markov blanket algorithm at an initial phase to reconstruct the local neighborhood around each node, and uses this knowledge to improve efficiency by focusing its attention in subsequent steps. We also presented a Monte Carlo version that has the advantages of faster execution speeds, "anytime" behavior, and potentially added reconstruction robustness due to multiple tests and Bayesian accumulation of evidence at the trade-off of completeness of the set of tests required. Simulation results demonstrate the reconstruction accuracy advantages of the algorithms presented here over hill-climbing methods. Additional results also show that the Monte Carlo version has a dramatic execution speed benefit over the plain one in cases where an assumption of bounded neighborhood may not hold, without significantly affecting the generalization error rate.

An avenue of further research in this direction, which would benefit the class of independence-based BN induction algorithms in general, would be the development of a "score" that measures the *overall* satisfaction of the independence relations implied by a given BN structure. This would centralize the multiple decisions that are presently done in all independence-based algorithms, making them prone to propagation of an early errors to later stages and therefore unstable. Combining all independence compliance in a single number would allow the algorithm to prefer one structure over another based on a single decision and hopefully tolerate little statistical confidence in one region of the network if the result of changing it would be disastrous in another region (due to, for example, the preservation of acyclicity constraints).

Second, we presented a probabilistic test of independence between two continuous variables. For this purpose, we proposed using the posterior probability of independence given the data set that takes into account histograms representing discretizations at several different resolutions. We showed that this is necessary by observing that independence of two variables across different discretization resolutions may vary substantially. We therefore developed a Bayesian approach that incorporates evidence from different resolutions, and in addition averages (by means of the Bayesian integral) over all possible boundary placements on the plane. By taking advantage of certain structure that exists in the space of discretizations, we were able to approximate the Bayesian integral, reducing the time that is required to estimate it from exponential in the size of the data set, to polynomial.

Further research in another direction would be the development of a *conditional* independence test for cases where the conditioning variables are also continuous. This would be an important advancement, since it would allow the use of any independence-based algorithm to be used in continuous domains without the use of a distributional assumption, as is the case presently (*e.g.* assuming all variables are Gaussian).

Our solution will benefit the determination of the structure of Bayesian networks in domains that contain any number of continuous, ordinal discrete and/or categorical attributes. In such domains, our test may be applied in a straightforward fashion, due to the fact that we employ discretization—therefore, the dependence of any kind of variables can be determined once the continuous ones have been discretized. (Some care needs to be taken when applying our algorithm to categorical variables, since their values are unordered and the "sweeping" part of the algorithm cannot be applied.) One additional benefit that may emerge from such an approach is the representation of the conditional probability distributions of a BN, if our test is being used for such a purpose. This is another topic of further research.

Another interesting direction of research is the development of quad-tree like discretization rather than axis-parallel boundaries that span the range of the variables involved. One can envision, for example, a discretization that detects ranges in the plane where the variables are locally approximately independent. Such a range may indicate the presence of a latent variable, explaining the local independence. This may be used to generate new, possibly meaningful features by combining existing ones in the domain.

Accelerating the asymptotic running time of our algorithm is important. Even though our method is polynomial in time, the exponent is the number of variables involved in the conditional test. This may be prohibitively expensive for large conditioning sets and/or data sets. A faster algorithm is required for those situations. We are currently exploring the approximation of the Bayesian integral by a Monte-Carlo version of our algorithm.

Further research is also needed to address problems with the test given a small number of data points in certain configurations, such as the one in Fig. 4.4. We are presently exploring the possibility of a test that recursively splits each axis using a "blind" split (rather than a search for one that maximizes the probability of dependence) on the median of each axis. The preliminary results are promising.

Finally, we proposed a *paradigm shift* in the approximate computation of count DataCubes: we propose to use a model of the data instead of the data themselves. Our approach, NetCube, uses the technology of

Bayesian networks to obtain the key advantage of large storage savings in situations where only approximate answers are needed. This makes feasible the computation of DataCubes for databases that were previously problematic using state-of-the-art methods such as bitmaps.

In our DataCube experiments we used only binary attributes. However, the concepts and implementation easily extend to multi-valued discrete data easily. NetCubes can also handle continuous attributes after bucketization *e.g.* “salary” could become a discrete attribute taking values “low” ($\leq$ 10,000), “medium” ($\geq$ 10,000 and $\leq$ 100,000) or “high” ($\geq$ 100,000). Chapter 4 describes a potential method of discretizing continuous attributes.

A subject of future research is the extension of the current system to the estimation of additional aggregate functions of the DataCube operator, in addition to counts. For example, knowing the probability distribution of a multi-valued attribute enables us to quickly estimate its average value. Other quantities such the minimum and maximum values can be read directly from representation of the Bayesian network.

Our approach theoretically lends itself easily to non-stationary distributions. Assume for example that new data are incorporated in the database periodically, *e.g.* a supermarket may append transaction data to the database at the end of each day. A data analyst may be interested in certain quantities on a per-day basis. In that case one possible solution is to compute a Bayesian network for each particular day. That network can answer queries for that day. However more often it is more useful to examine the behavior over broader time periods. For that purpose, the same approach will work: a query concerning several days, not necessarily consecutive, can be made to the corresponding (single-day) networks covering the time period of interest. The resulting counts can then be summed to obtain a count estimate for the entire time period. An open experimental problem is the method by which to decide on splitting the data into pieces: a day is a convenient short-term criterion but what about longer time periods (*e.g.* shopping seasons)? Also, what about “drifting” distributions, *e.g.* trends in consumer behavior that are slowly changing with time? Potential solutions include fixed-size splitting or detecting changes in the underlying distribution with time and invoking a split when the change is too great according to some reasonable metric. The selection of this metric as well as the identification of a detection algorithm that is sensitive to subtle changes is a subject of future research.

Appendix A

Computation of $\Pr(L \mid \Xi_m)$

The following derivation applies to the formulas in both steps 2 and 3 of the randomized version of the GS algorithm (Fig. 3.9). In the derivation below the following symbols are used:

- $L(X,Y)$ represents the event that variables X and Y are linked either through a direct edge (GS step 2) or by conditioning on a common descendant (step 3).
- $D_m(X,Y)$ for $m = 1,\ldots,M$ are events that X and Y are dependent conditioned on a set of variables $\mathbf{S}_m$, a subset of $\mathbf{B}(X) - \{Y\}$ (that includes a possible common descendant in step 3).
- $\Xi_m = \langle \xi_1, \xi_2, \ldots, \xi_m \rangle$ is the first m data sets, represented as a vector of data sets. The data points of each data set are assumed i.i.d. (independent and identically distributed).

In our notation we will omit the dependency on X and Y of the variables listed above for reasons of brevity *e.g.* $L(X,Y)$ will be written as L. We assume that $X \in \mathbf{B}(Y)$ and $Y \in \mathbf{B}(X)$, as the probabilities that are referred to below are computed only for such variables in the randomized GS algorithm.

We are interested in calculating the probability $\Pr(L \mid \Xi_m)$ in terms of $\Pr(D_m \mid \xi_m), i = 1,\ldots,M$, the only direct evidence we may elicit from each data set ξ_m. In other words, the results of the tests D_m are the only information we use from ξ_m, making them in effect the sufficient statistics of the data set vector Ξ_m as far as L is concerned. This is one of our assumptions.

We will formulate $\Pr(L \mid \Xi_m)$ in terms of $\Pr(L \mid \xi_m)$, the probability of L given a single test only, and $\Pr(L \mid \Xi_{m-1})$, the accumulated probability from previous data. We have

$$\Pr(L \mid \Xi_m) = \Pr(L \mid \xi_m, \Xi_{i-1}) = \frac{\Pr(\xi_m \mid L, \Xi_{m-1}) \Pr(L \mid \Xi_{m-1})}{\Pr(\xi_m \mid \Xi_{m-1})}$$

$$\Pr(\overline{L} \mid \Xi_m) = \Pr(\overline{L} \mid \xi_m, \Xi_{m-1}) = \frac{\Pr(\xi_m \mid \overline{L}, \Xi_{m-1}) \Pr(\overline{L} \mid \Xi_{m-1})}{\Pr(\xi_m \mid \Xi_{m-1})}.$$

From the two equations above, and using the i.i.d. assumption $\Pr(\xi_m \mid L, \Xi_{m-1}) = \Pr(\xi_m \mid L)$ and $\Pr(\xi_m \mid \overline{L}, \Xi_{m-1}) = \Pr(\xi_m \mid \overline{L})$, we get

$$\begin{aligned}
\frac{\Pr(L \mid \Xi_m)}{1 - \Pr(L \mid \Xi_m)} &= \frac{\Pr(\xi_m \mid L)}{\Pr(\xi_m \mid \overline{L})} \frac{\Pr(L \mid \Xi_{m-1})}{1 - \Pr(L \mid \Xi_{m-1})} \\
&= \frac{\frac{\Pr(L \mid \xi_m)\Pr(\xi_m)}{\Pr(L)}}{\frac{\Pr(\overline{L} \mid \xi_m)\Pr(\xi_m)}{\Pr(\overline{L})}} \frac{\Pr(L \mid \Xi_{m-1})}{1 - \Pr(L \mid \Xi_{m-1})} \\
&= \frac{\Pr(L \mid \xi_m)}{1 - \Pr(L \mid \xi_m)} \frac{1 - \Pr(L)}{\Pr(L)} \frac{\Pr(L \mid \Xi_{m-1})}{1 - \Pr(L \mid \Xi_{m-1})}. \qquad \text{(A.1)}
\end{aligned}$$

In the absence of any information, we assume that the probability that two variables are directly linked is equal to the probability that they are not *i.e.* $\Pr(L) = \frac{1}{2}$. The only remaining term is $\Pr(L \mid \xi_m)$:

$$\Pr(L \mid \xi_m) = \Pr(L \mid D_m \xi_m) \Pr(D_m \mid \xi_m) + \Pr(L \mid \overline{D}_m \xi_m) \Pr(\overline{D}_m \mid \xi_m). \qquad \text{(A.2)}$$

Since $\Pr(L \mid \overline{D}_m, \xi_m) = 0$ (if X and Y are not dependent they cannot possibly be linked), the second term at the sum above vanishes. Also, since ξ_m is only used to infer conditional dependence between X and Y (*i.e.* the value of D_m), knowledge of conditional dependence between X and Y (*i.e.* $D_m = 1$) makes ξ_m irrelevant in deducing the value of L. This implies $\Pr(L \mid D_m, \xi_m) = \Pr(L \mid D_m)$. Therefore,

$$\left.\begin{aligned}
\Pr(L \mid D_m, \xi_m) = \Pr(L \mid D_m) &= \tfrac{\Pr(D_m \mid L)\Pr(L)}{\Pr(D_m)} \\
\Pr(\overline{L} \mid D_m, \xi_m) = 1 - \Pr(L \mid D_m) &= \tfrac{\Pr(D_m \mid \overline{L})\Pr(\overline{L})}{\Pr(D_m)}
\end{aligned}\right\} \Rightarrow \Pr(L \mid D_m) = \frac{\Pr(D_m \mid L)}{\Pr(D_m \mid \overline{L}) + \Pr(D_m \mid L)}, \qquad \text{(A.3)}$$

since $\Pr(L) = \frac{1}{2}$. If we know there is a direct link between X and Y, the probability they are dependent is 1:

$$\Pr(D_m \mid L) = 1. \qquad \text{(A.4)}$$

If however we know that they are not directly linked, and given that each belongs to the blanket of the other, this probability depends on whether we include all common parents and no common children of them in the conditioning set $\mathbf{S}_m$. For that purpose we introduce two new events:

- A_m is the event that all common parents of X and Y are included in $\mathbf{S}_m$. We will abbreviate this as A in the formulas below.
- C_m is the event that some common children of X and Y are included in $\mathbf{S}_m$. This is abbreviated as C.

$$\begin{aligned}\Pr(D_m \mid \overline{L}) \quad = \quad & \Pr(D_m \mid AC\overline{L}) \Pr(AC \mid \overline{L}) + \\ & \Pr(D_m \mid A\overline{C}\overline{L}) \Pr(A\overline{C} \mid \overline{L}) + \\ & \Pr(D_m \mid \overline{A}C\overline{L}) \Pr(\overline{A}C \mid \overline{L}) + \\ & \Pr(D_m \mid \overline{A}\overline{C}\overline{L}) \Pr(\overline{A}\overline{C} \mid \overline{L}).\end{aligned} \tag{A.5}$$

Given that there is no direct link between X and Y (*i.e.* $\overline{L}$), the two variables are independent only in the case all common parents and no common children of X and Y are included in the conditioning set, namely if $A\overline{C}\overline{L}$ is true. Therefore

$$\begin{aligned}\Pr(D_m \mid AC\overline{L}) &= 1 \\ \Pr(D_m \mid A\overline{C}\overline{L}) &= 0 \\ \Pr(D_m \mid \overline{A}C\overline{L}) &= 1 \\ \Pr(D_m \mid \overline{A}\overline{C}\overline{L}) &= 1.\end{aligned} \tag{A.6}$$

Since the algorithm picks the members of the conditioning set entirely randomly we have

$$\begin{aligned}\Pr(AC \mid \overline{L}) = \Pr(A)\Pr(C) = \quad & \tfrac{1}{2^\alpha}(1 - \tfrac{1}{2^\beta}) \\ \Pr(A\overline{C} \mid \overline{L}) = \Pr(A)\Pr(\overline{C}) = \quad & \tfrac{1}{2^\alpha}\tfrac{1}{2^\beta} \\ \Pr(\overline{A}C \mid \overline{L}) = \Pr(\overline{A})\Pr(C) = \quad & (1 - \tfrac{1}{2^\alpha})(1 - \tfrac{1}{2^\beta}) \\ \Pr(\overline{A}\overline{C} \mid \overline{L}) = \Pr(\overline{A})\Pr(\overline{C}) = \quad & (1 - \tfrac{1}{2^\alpha})\tfrac{1}{2^\beta}\end{aligned} \tag{A.7}$$

where α is the number of common parents and β the number of common children of X and Y. Combining Eq. (A.5) with Eqs. (A.6) and (A.7) we get

$$\Pr(D_i \mid \overline{L}) = 1 - \frac{1}{2^{\alpha+\beta}}. \tag{A.8}$$

Since α and β are not known in advance, assuming (without loss of generality) that $|\mathbf{B}(X)| \leq |\mathbf{B}(Y)|$, we can approximate their sum with $|\mathbf{B}(X)| - 1 \stackrel{\text{def}}{=} |\mathbf{B}| - 1$ (we subtract 1 because Y is also a member of $\mathbf{B}$):

$$\Pr(D_m \mid \overline{L}) \approx 1 - \frac{1}{2^{|\mathbf{B}|-1}} \stackrel{\text{def}}{=} G. \tag{A.9}$$

G is a measure of how connected a node (in this case, X) is to the remaining variables. Its value ranges from 0 to 1 as $|\mathbf{B}|$ takes values from 1 to ∞. Combining Eqs. (A.2), (A.3), (A.4) and (A.9) (where we assume equality from now on), we get

$$\Pr(L \mid \xi_m) = \frac{\Pr(D_m \mid \xi_m)}{1 + G}. \tag{A.10}$$

This equation has an easily seen and intuitively appealing interpretation: when $|\mathbf{B}|$ is large ($G = 1$), the evidence of a test indicating dependence between X and Y conditioned on a randomly drawn subset $\mathbf{S}_m$ of the blanket of X bears little evidence of a direct link between them, since the probability that it includes all parents and no children of X is small and therefore the posterior probability of L is close to $\frac{1}{2}$. When $|\mathbf{B}|$ is 1

$(G = 0)$, however, the evidence for L is strong and the above probability becomes 1, since, in the absence of any common parents or children of X (or Y), any test indicating dependence is direct evidence of a link between X and Y.

Combining Eq. (A.1) with Eq. (A.10) and solving for $\Pr(L \mid \Xi_m)$, we obtain the final recursive formula, used in steps 3 and 4 of the randomized GS algorithm of Fig. 3.9:

$$\Pr(L \mid \Xi_m) = \frac{\Pr(L \mid \Xi_{m-1})\ \Pr(D_m \mid \xi_m)}{\Pr(L \mid \Xi_{m-1})\ \Pr(D_m \mid \xi_m) + (1 - \Pr(L \mid \Xi_{m-1})\ (G + 1 - \Pr(D_m \mid \xi_m))}.$$

Appendix B

MaxLikelihoodDAG is NP-complete

Theorem: Given a weighted directed graph $\mathcal{G} = \langle \mathcal{U}, \mathcal{E} \rangle$ such that $\forall X,Y \in \mathcal{U}$, if $(X,Y) \in \mathcal{E}$ then $(Y,X) \in \mathcal{E}$, and $w(\cdot \to \cdot)$ is a weight function for each directed edge, the problem of determining the acyclic graph of the maximum product of edge weights (call it *MaxLikelihoodDAG*) is NP-complete.

Proof: We reduce from the minimum feedback arc set problem (*MFAS*). The *MFAS* problem is about finding the smallest set of edges in a directed graph $\mathcal{G}_{MFAS} = \langle \mathcal{U}_{MFAS}, \mathcal{E}_{MFAS} \rangle$ whose removal will make the resulting graph undirected. To reduce it to *MaxLikelihoodDAG* we define $\mathcal{G} = \langle \mathcal{U}, \mathcal{E} \rangle$ with weight function w as follows

$$\mathcal{U} = \mathcal{U}_{MFAS}$$

$$\mathcal{E} = \mathcal{E}_{MFAS} \cup \{(Y,X) \mid (X,Y) \in \mathcal{E}_{MFAS}\}$$

$$w(X \to Y) = \begin{cases} 2 & \text{if } (X,Y) \in \mathcal{E}_{MFAS} \\ 1 & \text{if } (X,Y) \notin \mathcal{E}_{MFAS} \end{cases}$$

Calling *MaxLikelihoodDAG* will return a subgraph such that $\prod_{(X,Y)\in\mathcal{E}} w(X \to Y)$ is maximum or, equivalently, $\sum_{(X,Y)\in\mathcal{E}} \log_2 w(X \to Y)$ is maximum. Removing the edges that have $\log_2 w(X \to Y) = 0$ gives us a solution to the *MFAS* problem. This is because of two reasons. Firstly, since removing edges cannot introduce a cycle and since all remaining edges in the solution are members of $\mathcal{E}_{MFAS}$, it is a legal solution to the feedback arc problem. Secondly, because it contains a maximum number of edges it is a solution to the minimum feedback arc problem. We prove this claim by contradiction: suppose that there exists an acyclic subgraph of $\mathcal{G}_{MFAS}$ that has a greater number of edges. Then by using the edge weight assignment described above, we can produce a better solution (greater weight product) to *MaxLikelihoodDAG*, which is a contradiction.

This proves NP-hardness of *MaxLikelihoodDAG*. NP-completeness is due to the polynomial-time conversion (in the size of the graph) from the *MaxLikelihoodDAG* solution to a *MFAS* one, which simply involves the omission of the edge weights.

Bibliography

J. M. Agosta. The structure of Bayes networks for visual recognition. In *Uncertainty in Artificial Intelligence*, pages 397–405. Elsevier Science Publishers B.V. (North-Holland), 1990.

A. Agresti. *Categorical Data Analysis*. John Wiley and Sons, 1990.

D. Barbará. Quasi-cubes: A space-efficient way to support approximate multidimensional databases. Technical report, ISE Dept., George Mason University, 1998.

R. J. Bayardo Jr. and R. Agrawal. Mining the most interesting rules. In *Proc. of the Fifth ACM SIGKDD International conference on Knowledge and Discovery*, pages 145–154, 1999.

K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and Iceberg CUBEs. In *Proceedings of the 25th VLDB Conference*, pages 359–370, 1999.

R. Bowden and D. Turkington. *Instrumental Variables*. Cambridge University Press, New York, 1984.

W. Buntine. Theory refinement on Bayesian networks. In B. D'Ambrosio, P. Smets, and P. P. Bonissone, editors, *Uncertainty in Artificial Intelligence (UAI)*, pages 52–60, Los Angeles, CA, 1991.

W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London, 1993.

C.-Y. Chan and Y. Ioannidis. Hierarchical cubes for range-sum queries. In *Proceedings of the 25th VLDB Conference*, pages 675–686, 1999.

K.-C. Chang and R. Fung. Symbolic probabilistic inference with continuous variables. In *Uncertainty in Artificial Intelligence (UAI)*, pages 77–85. Morgan Kaufmann, 1991.

R. M. Chavez and G. F. Cooper. A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, 20:661–685, 1990.

J. Cheng, D. A. Bell, and W. Liu. An algorithm for Bayesian network construction from data. In *Artificial Intelligence and Statistics*, 1997.

D. Chickering. Learning equivalence classes of Bayesian-network structures. In *Proceedings of the Twelfth Conference in Artificial Intelligence*, Portland, OR, 1996. Morgan Kaufmann.

D. M. Chickering. Learning Bayesian networks is NP-complete. In *Proceedings of AI and Statistics*, 1995.

D. M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, November 2002.

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

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

A. Darwiche. Conditioning methods for exact and approximate inference in causal networks. In *Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann, 1995.

S. Davies and A. Moore. Bayesian networks for lossless dataset compression. In *Conference on Knowledge Discovery in Databases (KDD)*, 1999.

S. Davies and A. Moore. Mix-nets: Factored mixtures of gaussians in Bayesian networks with mixed continuous and discrete variables. Technical Report CMU-CS-00-119, Carnegie Mellon University, School of Computer Science, April 2000.

N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *Uncertainty in Artificial Intelligence (UAI)*, pages 165–174, 1997.

N. Friedman, M. Goldszmidt, and T. J. Lee. Bayesian network classification with continuous attributes: Getting the best of both discretization and parametric fitting. In *Proc. Fifteenth International Conference on Machine Learning (ICML)*, pages 179–187, 1998.

R. Fung and K. C. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In L. Kanal, T. Levitt, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence*. Elsevier Science Publishers B.V. (North-Holland), 1989.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. CRC Press, 1995.

L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic relational models. In *Proceedings of the ACM SIGMOD (Special Interest Group on Management of Data) Conference*, 2001.

J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *International Conference on Data Engineering*, 1996.

V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, March 1995.

D. Heckerman and D. Geiger. Likelihoods and parameter priors for Bayesian networks. Technical Report MSR-TR-95-54, MicroSoft Research, Redmond, WA, November 1995.

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

D. Heckerman, A. Mamdani, and M. P. Wellman. Real-world applications of Bayesian networks. *Communications of the ACM*, 38(3):24–30, 1995b.

M. Henrion. Propagation of uncertainty by probabilistic logic sampling in Bayes' networks. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence 2*. Elsevier Science Publishers B.V. (North-Holland), 1988.

E. H. Herskovits and G. F. Cooper. Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. In *Uncertainty in Artificial Intelligence (UAI)*, 1990.

C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 11:1–158, 1994.

Y. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query answers. In *Proceedings of the 25th VLDB Conference*, pages 174–185, 1999.

H. Jeffreys. *Theory of Probability*. The International Series of Monographs on Physics. Oxford At The Clarendon Press, 3rd edition, 1961.

F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

T. Johnson. Performance measurements of compressed bitmap indices. In *Proceedings of the 25th VLDB Conference*, pages 278–289, 1999.

M. Jünger. *Polyhedral combinatorics and the acyclic subdigraph problem*. Heldermann, Berlin, 1985.

R. Kass and A. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90:773–795, 1995.

R. Kass, L. Tierney, and J. Kadane. Asymptotics in Bayesian computations. In J. Bernardo, M. DeGroot, D. Lindley, and A. Smith, editors, *Bayesian Statistics 3*, pages 261–278. Oxford University Press, 1988.

W. Lam and F. Bacchus. Learning Bayesian belief networks: an approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994a.

W. Lam and F. Bacchus. Using new data to refine a Bayesian network. In *Uncertainty in Artificial Intelligence (UAI)*, 1994b.

P. Larranaga, M. Poza, Y. Yurramendi, R. H. Murga, and C. M. H. Kuijpers. Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Journal on Pattern Analysis and Machine Intelligence (PAMI)*, 18(9):912–926, 1996.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal Royal Statistical Society, Series B*, 50(2):157–224, 1988.

J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, pages 205–214, 1999.

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. Technical Report CMU-CS-99-134, Carnegie Mellon University, School of Computer Science, August 1999.

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 505–511. MIT Press, 2000.

C. Meek. *Graphical Models: Selecting causal and statistical models*. PhD thesis, Carnegie Mellon University, Dept. of Philosophy, 1997.

N. Megiddo and R. Srikant. Discovering predictive association rules. In *Proc. of the 4th Int'l Conference on Knowledge Discovery in Databases and Data Mining*, New York, USA, 1998.

S. Monti and G. Cooper. Learning hybrid Bayesian networks from data. In M. Jordan, editor, *Learning and Inference in Graphical Models*. Kluwer Academic Publishers, 1998a.

S. Monti and G. F. Cooper. A multivariate discretization method for learning Bayesian networks from mixed data. In *Uncertainty in Artificial Intelligence (UAI)*, pages 404–413, Madison, WI, 1998b. Morgan Kaufmann.

P. M. Murphy and D. W. Aha. *UCI repository of machine learning databases*. University of California, Department of Information and Computer Science, Irvine, CA, 1994. [`http://www.ics.uci.edu/~mlearn/MLRepository.html`].

J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32: 245–257, 1987.

J. Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–709, 1995.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2nd Ed., 1997.

J. Pearl and T. S. Verma. A theory of inferred causation. In J. F. Allen, R. Fikes, and E. Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 441–452, San Mateo, California, 1991. Morgan Kaufmann.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd Ed., 1992.

G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 222–228. Elsevier Science Publishers B.V. (North-Holland), 1989.

J. Rissanen. Stochastic complexity (with discussion). *Journal of the Royal Statistical Society, Series B*, 49: 223–239 and 253–265, 1987.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

R. D. Schachter. Evidence absorption and propagation through evidence reversals. In *Uncertainty in Artificial Intelligence (UAI)*, pages 173–190, San Francisco, 1990. Elsevier Science Publishers B.V. (North-Holland).

R. D. Schachter, B. D'Ambrosio, and B. A. Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth Conference in Artificial Intelligence*, pages 126–131. MIT Press, 1990.

R. D. Schachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence (UAI)*. Elsevier Science Publishers B.V. (North-Holland), 1989.

G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proceedings of the 24th VLDB Conference*, pages 594–605, 1998.

D. Spiegelhalter and S. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.

P. Spirtes. An anytime algorithm for causal inference. In *Proc. of International Workshop on Artificial Intelligence and Statistics*, 2001.

P. Spirtes, G. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Springer-Verlag, New York, 1993.

H. J. Suermondt and G. F. Cooper. Probabilistic inference in multiply connected networks using loop cutsets. *International Journal of Approximate Reasoning*, 4:283–306, 1990.

J. Suzuki. Learning Bayesian belief networks based on the MDL principle: an efficient algorithm using the branch and bound technique. In *Proceedings of the International Conference on Machine Learning*, Bally, Italy, 1996.

T. S. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Uncertainty in Artificial Intelligence (UAI)*, pages 220–227, San Francisco, 1990. Elsevier Science Publishers B.V. (North-Holland).

J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 193–204, 1999.

M.-C. Wu. Query optimization for selections using bitmaps. In *Proceedings of the 25th VLDB Conference*, pages 227–238, 1999.