INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Learning in Belief Networks and its Application to Distributed Databases

Messaouda Ouerd

A Thesis

Submitted to the School of Graduate Studies and Research

of the University of Ottawa in partial fulfillment of the requirements

for the degree of Doctor of Philosophy in Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa

Ottawa, Ontario, Canada

March 2000

Abstract

In this thesis we study the problem of learning in belief networks and its application to caching data with repeated read-only accesses in distributed databases.

Bayesian Belief Networks (BBNs) have been studied in the literature, and two classes of techniques for constructing BBNs from distributions have been studied. These schemes are methods based on probabilistic-graph models, and Bayesian methods for learning Bayesian networks. In this thesis we first consider methods to build tree structures and use these trees as a basis to build a richer structure, namely a polytree graph. We study the problem of traversing the tree and present a depth first search traversal of the tree in order to orient it so as to yield the polytree. The algorithm to yield the above polytrees uses independence tests between two random variables to detect multiple parents of a given node in the tree structure. Consequently we investigate the use of various independence tests to infer independence of random variables encountered in real-life data. We also present formal techniques to generate random distributions obeying polytree dependence models.

The thesis also develops machine learning schemes to detect sequences of repeated queries to remote databases. The answers to these queries (tables) from remote servers are retrieved only once and cached locally in memory. Subsequent access to the same data or sequence of data is faster as there is no need to re-fetch it over the network. The learning algorithms we present are based on constructing polytree structures from a set of queries. Once constructed, such networks can provide insight into probabilistic dependencies that exist among the queries and thus enhance distributed query optimization.

To my children: Sarah, Sofiane and Yazid and my husband Arab

Acknowledgments

I would like to put on record my gratitude to my supervisors Dr. Stan Matwin and Dr. John Oommen for their continued support during my Doctoral studies. I was fortunate to have Dr. Matwin as my supervisor at the University of Ottawa. I am grateful to him for taking me on as his student and for supervising the practical and applied aspects of the thesis. I am especially grateful to Prof. Oommen, my supervisor who worked out of Carleton University. I am grateful to him for supervising the theoretical aspects of the problems, his patience, motivation and continuous enthusiasm. He reads the various drafts of the thesis and spent countless hours with me, for which I stand in debt.

I am grateful to Dr. Rob Holte for going through the thesis with a fine comb. I also thank Dr. Joel Martin, Dr. Franz Oppacher and Dr. Mario Marchand for their critical and helpful comments, which improved the readability of the thesis. I am also grateful to Dr. Douglas King for providing me with the data and with much insight with regard to the problem studied in Chapter 6.

I would like to thank the Canadian International Development Agency and the University of Ottawa for their financial support.

My husband Arab deserves the most thanks for his infinite encouragement, moral support, patience and understanding.

Finally, I thank my parents for encouraging me throughout the years of my studies.

Table of Contents:

List of Figures

List of Tables

Chapter 1

Introduction

The introduction of the digital computer totally revolutionized the way we solved complex problems. Computers have been particularly useful in the area of efficiently storing, retrieving and permitting fast and intelligent access to data. Tasks which previously required human intelligence can now be accomplished as the result of the intense research in Artificial Intelligence (AI). Advances have been made in applying AI techniques to problems found intractable or difficult for traditional computation.

The goal of AI is to provide a computational model of intelligent behavior, the most important aspect being commonsense reasoning. In AI, the use of probability theory shows how belief should vary when we are given partial or uncertain information. With the fact that commonsense reasoning always applies to incomplete information, we could then expect the two disciplines to share language, goals and techniques.

The recent and promising results in solving AI problems using probability theory have encouraged the research in the development of new and innovative probabilistic schemes. Probability Theory views the world as a set of random variables $X_1,...,X_N$ each of which has a domain of possible values. Unfortunately an explicit description of the joint distribution requires a number of parameters that is exponential in N, the number of variables.

Over the past two decades the advantages of coherent probabilistic schemes for representing uncertainty in AI has become more widely recognized. Consequently, there has been increasing interest in the use of structures such as *Bayesian Belief Networks* (BBN). Indeed, the importance of building probabilistic (and more recently, database) dependency structures from empirical observations is well acknowledged. It has emerged to be a dominant area in machine learning. In recent years, the theory of BBNs has provided powerful techniques for representing and manipulating knowledge in the form

of causal Directed Acyclic Graphs (DAGs), which, in turn, have been fundamental to describing causal influences among uncertain random variables [Pea88]. One attractive feature of BBNs is that they allow a convenient way to represent causal knowledge. In the terminology of conditional independence, this is expressed by stating that a set of nodes is conditionally independent of all their non-descendants given their parents. Prior knowledge can thus be combined with observed data to determine the final probability of a hypothesis.

The fundamental problem concerning the synthesis of a Bayesian Belief Network (BBN) involves capturing random variable dependencies which, in turn, focuses on constructing a graph that captures node dependencies. The idea is to find the graph that adequately models the accurate representation of a given probability distribution. Since the number of possible structures grows exponentially with the number of the nodes of the graph, an exhaustive enumeration of all network structures is not feasible in most domains. Thus, since the search space is enormous, most algorithms use heuristics to render search tractable.

Robinson [Rob77] derived the following formula for determining the number of possible belief network structures that contain n nodes. It is a recursive function f(n) where :

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} \; 2^{i(n-1)} \, f(n-i) \text{ and}$$

$$f(0) = 0.$$

Thus, for example if n = 5, the number of possible BBNs is 29,000, and for n = 10 the number of possible BBNs is $4.2 * 10^{18}$!! Indeed, Chickering and Heckerman [CH94] and Cooper [Coo87] showed that the synthesis (construction) of the BBN is NP-hard. Singly connected networks can, however, be efficiently solved in linear time [Cha91]. More recently, AI researchers from various schools have attempted to develop methods for *learning* Bayesian networks, using methods which are themselves Bayesian methods [CH92], or non-Bayesian methods [Pea88]. Bayesian learning methods determine the most probable hypothesis $h \in H$ from some space H that explains the observed training data D given that data D and an initial knowledge about the prior probabilities of the

various hypotheses in H, called a *maximum a posteriori* (MAP) hypothesis. It is easy to see that an exhaustive search to determine the MAP hypothesis is impractical. Most algorithms that learn the BBNs using the Bayesian learning methods exploit heuristics to reduce the search space. These algorithms perform a greedy search that trades network complexity off for accuracy over the training data for learning the structure of the Bayesian network. A survey of these methods is given in Chapter 2. The probabilistic methods focus on finding the most likely structure, implicitly assuming that there is a true underlying structure. This constitutes the bias for these methods. If the underlying probability distribution demonstrates the given structure dependence, the learning methods asymptotically reproduce the underlying structure. If the underlying probability distribution is not of that structure dependence, these methods produce an approximation to these probabilities distributions. In this thesis we opted for using the polytree based approximation to our underlying distribution. In our learning scheme this restriction forms the restriction bias [Mit97].

The topics studied in this thesis fall under two categories. The first is the knowledge representation problem, which is from the overall field of machine learning, and in this area we propose algorithms for the representation of knowledge in the form of polytrees. The second uses the former solutions to problems related to query optimization in distributed databases. The intention of this introductory chapter is to give the reader a global view of the nature of the problems that we have tackled and the various solutions proposed. This introductory chapter concludes with an outline of the contribution of this thesis.

1.1 Knowledge Representation

1.1.1 Approximating Probabilities

Many of the stochastic models we would like to work with are intractable. The challenge is to find general-purpose, tractable approximation algorithms for reasoning with these elegant and expressive stochastic models. This is because large machine learning problems are beginning to arise in numerous application domains. For example, these problems arise in distributed database applications where there can be millions of

transactions every day and where it is desirable to have machine-learning algorithms that can improve system performance.

With a little insight it can be seen that a fundamental problem encountered in all such situations is that of finding a good approximation for the underlying distribution P(X), where X is a vector representing the values of features in the problem domain. The problem (a subproblem in knowledge representation) that we study is of approximating the probability distribution P(X) by a well-defined and easily computable density function $P_a(X)$. Indeed it is impractical to store all estimates of the joint function P(X) for all possible values of the vector X. Our goal is to build a probabilistic network from the distribution of the data, which adequately represents the data. Once constructed, such a network can provide insight into probabilistic dependencies that exist among the variables.

In order to measure the "goodness" of the approximation, an information theoretic measure is given by the Kullback-Leibler cross-entropy measures to compare joint probability distributions. Chow and Liu [CL68] used this measure to approximate discrete distributions by collecting the entire first and second order marginals. They derived a relationship between the measure of closeness between the probabilities and the measure of independence between all the pairs of the variables. A maximum weight spanning tree called the Chow tree was built using the information measure between the variables forming the nodes of the tree. Tree structures are known to be efficient for extracting causal knowledge [Pea88].

Subsequent work by Rebane and Pearl [RP87] use the Chow algorithm as the basis of an algorithm which builds a polytree (singly connected network) from a probability distribution. Noteworthy are the results of Srivinas *et al.*, [SRA90] who worked with independence and the recent results of Dasgupta [Das99] which explicitly specifies the complexity of the underlying problem. Friedman [Fri98] has also worked in the area and has modified the traditional EM algorithm to devise the "Structural" EM algorithm [Fri98] to learn BBNs, and also demonstrated how one can learn Bayesian Networks from massive data sets using the "Sparse Candidate Algorithm" in [FNP99].

The reader should observe that polytrees are much richer structures since they use higher order probability densities. This algorithm orients the Chow tree (the skeleton of the polytree) assuming the availability of an independence test on any multiple parent nodes. In Chapter 3 we show how this polytree can be constructed by implementing a depth first search traversal procedure to orient the skeleton tree. In Chapter 5 we investigate the use of different methods to obtain independence tests when the equality $P(X,Y) = P(X)\ P(Y)$ which exactly characterizes the statistical independence of two random variables X and Y *fails* because of the real-life data or the sample data. In Chapter 4, we described particularly, the problem of how could we obtain a valid *distribution* that the polytree structure will permit in order to derive samples following that polytree structure. This data generation process will allow us to create synthetic data in order to test the algorithms we developed in the previous chapters.

1.2 Query Optimization in Distributed Systems and Network Caching

Computer technology has forced its way into almost every facet of human life in the developed world, and this has been driven by the availability of high-technology resources and solutions. This technology revolution has created an increasing need to decentralize, or distribute, the information processing task. Computer communications is a byproduct of this need, and has now evolved into a highly complex technology [Heb92].

The significance of Wide Area Networks (WANs) has increased with the population of computers connected to them, the range of software packages supporting their use has also correspondingly grown. With the vast amount of knowledge that is currently available to a computer system analyst, he faces an ensemble of problems. First of all, he encounters the problem of efficiently representing the data. The data used by a number of applications[1] may be distributed to a number of processing sites. Processing is concerned with a number of aspects, including collection, storage and manipulation of information. The distribution implies repeated access to the data stored in remote

[1] The term application refers to an orchestrated collection of information processing capabilities provided to some consumer which can be a person or an "application itself" [Heb 92].

databases on top of the query-processing task. The problems encountered in query processing in databases take on additional complexity in a distributed environment because of the communication costs of query accesses.

One major motivation for the work in this thesis is to increase performance in systems involving distributed databases. As the usage of WANs increases, the need for enhancing the systems' performance becomes apparent. In many distributed applications the system makes repeated access to the same remote data. We thus notice an increase in query accesses for fetching the data. The traditional way of dealing with query optimization is to find an execution strategy, which is optimal, or close to optimal considering the fact that communication cost is involved. In general, query processing methods for distributed database depends mostly on communication cost. Indeed, the communication cost is probably the primary and dominating factor that determines distributed query optimization. It is crucial in a sense that if this factor can be improved, the overall performance of the whole distributed database system can be enhanced. In order to minimize the communication cost, most methods consider the following factors: transportation cost, access methods, order of performing joins, method of executing the joins and determining the locations where the joins are done. Consequently, research in this field has involved strategies such as database knowledge discovery, query reformulation, etc.

In this thesis we approach this problem of query optimization in distributed databases in a completely different way; one which is inspired by the fact that communication time is the main factor considered by the database management system in distributed databases. In our approach we reduce the communication time by minimizing the number of queries made to remote databases by caching the repeated queries, and thus avoiding the re-fetching of the same queries. Indeed, caching data at the local user station allows subsequent access to that data to be local and thus improves the performance of the system. To achieve this we make decisions on caching by learning the workflow of the data retrieved by a certain application. Anticipated caching is also achieved by learning the workflow of the data given a trace of queries made to the databases. These queries are typically captured during certain periods of time for learning. Using learning

principles we show that we can build causal structures, primarily because causality is important to our caching process since it allows the anticipated caching mechanism. In fact, when accessing a query A and fetching data for it, we can also fetch data for query B if we know or if we have learned that query B has a high probability of following query A. This leads us to the possibilities of building causal structures which allow inference of causality.

The main objective of the application considered in this work is to improve the performance of systems in distributed databases. The heavy workload on the servers is due to the number of accesses which in turn, is related to the communication time. Indeed, this problem generally has to do with the general area of query optimization.

Our solution fits into the Open Distributed Processing (ODP) model which covers the dynamic binding of clients to service end-points. In this model, the client invokes a service, which communicates with a trader. The trader dynamically determines the best service to process the query, sets up the bindings and calls the server to process the request. The trader directs client requests to the most appropriate service. The binding delivers the request and the server processes it. We make decisions on caching (the cache management) by using results obtained from our inductive learning algorithms.

In the inductive learning scheme, the training data, which we use in this application, is a trace of queries made to the distributed database(s). This trace of queries is not used directly by our learning algorithms, but first parsed into more structured queries and generalized. Since we deal with applications that make repeated access to the same data in distributed databases, these queries are mainly select statements modeled as though they are written in the Structured Query Language (SQL). A select statement retrieves columns of data from database tables. After parsing the training data we proceed to the generalization process. Two or more consecutive queries having equal columns and equal tables in their select form are generalized into one single query. The idea behind this generalization is to compress the number of queries and thus reduces the number of accesses to the database by grouping queries that access the same columns of data stored in the same tables into one single query. Consequently we perform fewer accesses to the database which contains these tables. After the generalization process, we represent the

generalized queries as a binary vector X. The values of X are used by the learning algorithms described in this thesis to build the probabilistic network, which in turn will be used to infer query dependencies.

1.3 Organization of this Thesis and Its Contributions

The organization of this document is as follows. Chapter 2 describes some related work concerned with building probabilistic networks from data. The literature survey in this case is fairly comprehensive and generally speaking, reports the state of the art. This includes methods based on probabilistic graph models and Bayesian methods to build Bayesian belief networks. In Chapter 3 we consider the problem of orienting the tree obtained by the Chow algorithm to obtain a belief network as suggested by the Rebane and Pearl [RP87] algorithm. We consider a depth first search to orient the skeleton of the polytree. This orientation is based on decision of independence tests of every pair of nodes. Chapter 5 describes in detail how these tests can be obtained considering the fact that in any application which deals with real-life data, it may be impossible to obtain pairwise independence decisions.

In Chapter 4, we develop an algorithm for generating samples following an underlying structure. We consider the case when the entire first order probabilities are given and we create input data for the algorithms we develop in the previous chapters. Finally, in Chapter 6, we consider the application domain of our thesis. We investigate a new approach to optimizing distributed queries and introduce the concept of caching the repeated queries. It describes the details of processing the queries made to the distributed databases. Chapter 6 deals with the problem of using the polytree representation in our real-life application to optimize query processing in distributed databases. Chapter 7 concludes the thesis and briefly discusses the future work.

The thesis contains many new and novel results. To permit the reader to have an overall survey of our specific contributions, they are listed below. In Chapter 3 we presented new results to:

- Approximate distributions by a polytree based distributions.

- Apply a Depth-First Search (DFS) to causal basins in order to orient the polytree structure by presenting a formal method to traverse the tree.
- Prove that the DFS does indeed orient the tree and we have given the number of independence tests required to complete the orientation.
- All of the algorithms in this chapter were implemented and were justified by experimental results.

In Chapter 4 the new results, which we presented, include:

- An algorithm, which, generates samples from an underlying structure (DAGs) when the first order probabilities are given or the conditionals are given.
- An algorithm to generate the joint probabilities and therefore generated the samples from an underlying polytree based distribution.
- Again, these algorithms were formally proven to be correct and were both implemented and tested.

In Chapter 5 the contributions of the thesis were:

- Algorithms for learning polytrees when the joint or marginal probability distributions are not explicitly given but only samples are presented to the learner using the correlation coefficients.
- An implementation of the algorithm for multi-feature variables.
- A testing of the scheme for a real-life application (the Alarm network) for which we ran the polytree algorithms on the provided data.

In Chapter 6 we applied the concepts studied in knowledge representation to query optimization in distributed databases. In particular, we studied a real-life application for which we collected the data, parsed the queries, generalized them and built the first and second order marginals. Using these we built the Chow tree and polytree structures. All the prototype algorithms of the previous chapters were extended for this application domain, and consequently, the experimental results presented represent what can be expected by using the algorithms in a real-life scenario.

Chapter 2

Building Bayesian Networks

2.1 Introduction: Why Bayesian Networks?

The main thrust of this chapter is to review some of the existing algorithms that construct belief network structures automatically from empirical observations that have been obtained by taking "measurements" on discrete or continuous joint probability density functions.

BBN structures formally encode the joint probability distribution function. The problem with using rigorous probability theory to completely encode the data is that the complete specification of a probability distribution requires storing exponential number of values. To make this concrete, suppose we have a domain of 40 Boolean random variables (n = 40) where each variable is directly influenced by at most k = 6 other random variables. If these variables represent the nodes of a BBN, then specifying the conditional probabilities for a single node requires $O(2^k) = 64$ numbers, and thus $n*2^k = 2560$ numbers will be required for the complete network. To specify the complete joint distribution in this case, one needs to specify $2^n - 1 = 1{,}099{,}511{,}627{,}775$ numbers, which is clearly infeasible.

Since it completely represents the domain, a BBN is far more compact than the full joint distribution. In fact, treating the underlying distribution as a large table instead of a composition of several conditional probabilities might be very inefficient both in space and time requirements.

In this chapter we describe two classes of techniques for the induction of Bayesian networks from data; methods based on probabilistic-graph[2] models (section 2.3 through 2.6) and methods using a Bayesian learning approach (section 2.7). The probabilistic

[2] These models are essentially maximum likelihood models

methods for learning BBNs focus on finding the most likely structure, implicitly assuming that there is a true underlying structure. The Bayesian methods for learning BBNs search the space of possible network structures to yield the network, which aims to maximize the relative *a posteriori* probability.

2.2 Probabilistic Dependence: Background and Definitions

We consider a finite set U of discrete random variables, where each variable $X \in U$ may take values from a finite domain D. Capitals letters will be used for variable names (X, Y, Z, ...), and lowercase letters (x, y, z, ...) will be used for specific values taken by the variables X, Y, Z,..... respectively. We also denote by $\underline{X}$ the vector $\underline{X} = [X_1, X_2, ..., X_N]^T$, and by X a set of variables.

A dependency model is a set of statements of the form "X is independent of Y given Z" denoted as I(X, Z, Y) where X, Y and Z are disjoints sets of variables in the model [Pea88]. The notation I(X, Z, Y) will be used to denote the conditional independence of X and Y given Z. I(X, ∅, Y) will be used to denote unconditional independence (or marginal independence) of X and Y.

If X, Y and Z are three disjoint subsets of variables of a distribution P, then X and Y are said to be conditionally independent given Z, denoted I(X, Z, Y) if and only if

$$P[X = x, Y = y | Z = z] = [P(X = x | Z = z) * P(Y = y | Z = z)]$$

for all possible values x, y and z of X, Y and Z respectively. Given the set of predicates described above, the problem is to efficiently construct a BBN to represent the dependency model.

According to [Pea88] a necessary and sufficient condition for a DAG D to be a BBN of a probability distribution P is that each variable X be conditionally independent of all its non-descendants given its parents $\prod_X$, and that no proper subset of $\prod_X$ satisfy this condition. In the construction of Bayesian networks, the role played by the set of parents $\prod_X$ of a variable X is that they screen this variable X from the

influence of all other predecessors of X [Pea88]. This graphical criterion, called d-separation, associates the topology of the network with independencies encoded in the underlying distribution, and thus identifies conditional independencies in the BBN. The statistical meaning of any causal model can be described completely by its stratified protocol, which is a list of independence statements each asserting that a variable is independent of its non-descendants given its parents [VP91]. Geiger *et al.* [GVP90] developed an efficient algorithm that detects all independence implied by the topology of a Bayesian network. This algorithm is based on d-separation and runs in time O(|E|) where E is the number of edges in the network. To formalize the problem we first formally define the concept of d-separation.

Definition 2.1 [RP87]: A *trail* in a DAG is a sequence of links that forms a path in the underlying undirected graph. A trail is said to contain the nodes adjacent to its links.

Definition 2.2 [RP87]: A node b is called a *head-to-head* node with respect to a trail if there are two consecutive links a→b and b←c on t. A node that starts or ends a trail t is not a head-to-head node with respect to t.

Definition 2.3 [RP87]: Given three disjoint node sets X, Y and Z in a directed acyclic graph, X is said to be d-separated from Y by Z (denoted $< X \mid Z \mid Y >$), if and only if there exists no trail t between a node in X and a node in Y along which

(i) every head-to-head node (wrt t) either is or has a descendant in Z and,

(ii) every node that has an arrow along the trail t is outside Z.

A trail satisfying the two conditions above is said to be *active*. Otherwise, it is said to be *blocked* by Z.

Definition 2.4 [Pea88]: A DAG D is called an *Independency Map* (I-map) if every d-separation in the graph implies the corresponding independence in the underlying dependency model M, i.e.,

$$< X \mid Z \mid Y >_D \Rightarrow I(X, Z, Y)_M.$$

A DAG D is a *minimal I-map* of a dependency model if none of its arrows can be deleted without destroying its I-mapness. A BBN is a minimal I-map of a dependency model.

Definition 2.5 [Pea88]: A DAG D is called a *Dependency Map* (D-map) of a dependency model M if every non d-separation in the DAG corresponds to a non-independence in the underlying dependency model, i.e.,

$$I(X, Z, Y)_M \Rightarrow <X | Z | Y>_D$$

Definition 2.6 [Pea88]: A DAG D is said to be a *perfect map* of the dependency model if it is both an I-map and a D-map.

Observe that in such a case D represents all the dependencies and independencies of the model. For many dependency models, there is no DAG that is a perfect map of the dependency model. If a DAG structure is used to represent a dependency model, the structure that exhibits as many of the model's independencies as possible should be considered, and such a DAG is a minimal I-map of the dependency model [SRA90].

The task of finding a DAG which is a minimal edge I-map of a given probability distribution P and which yields a solution to the problem of building a BBN for a given probabilistic model and a node ordering d of the variables of P was solved in [PV87] and in [VP88]. The algorithm is formally given below using the notation that M is a dependency model and $d = X_1, X_2, ..., X_N$ is an ordering defined on the variables of the model. Also, we let $Pred(X_i)$ denotes all the predecessors of the variable X_i in the ordering d.

Algorithm Verma/ Pearl
Input: Conditional independent statements for every variable X_i.
Output: A minimal I-map BBN
Method:
Begin
For every node X_i **Do**
Pred(X_i) = { X_1, X_2, ..., X_{i-1} }
Find a minimal set of predecessors Parents(X_i) such that
Parents(X_i) $\subseteq$ Pred(X_i) and I(X_i, Parents(X_i), Pred(X_i) - Parents(X_i)) is true.
Assign a direct link from every variable in Parents(X_i) to X_i.
EndFor
End Algorithm Verma/ Pearl

The DAG created by designating the nodes corresponding to the variables in Parents(X_i) as the parents of the node X_i for all variables X_i i=1,N is called a minimal I-map of the probabilistic model. Deleting any edge from this DAG will destroy its I-mapness. We would like to emphasize that with this algorithm, the form of the resulting Bayesian network depends on the ordering of the variables of P because the ordering of the nodes can make a difference in the number of links in the resulting Bayesian network. A consequence of this observation is that the DAG generated by this algorithm is guaranteed to be a minimal I-map, which may not be sparse. A probability model may have many DAGs each corresponding to a different ordering of its variables. A structure representing a probabilistic model is useful if it is computationally tractable, which means that it is efficient both in its time and space requirements.

2.3 Dependence Trees and Cross- Entropy $I_f(.,.)$.

In the previous section we presented the Verma/Pearl algorithm for representing distributions. Using DAGS, we shall proceed to consider simplified graph representations using trees, polytrees etc. One of the pioneering results in this field was the efficient computation of *approximate* distributions, where the approximation is obtained using the tree representation, which, in turn, is very important, when we are interested in *causal* knowledge. The pioneering results in this area are due to Chow and Liu [CL68], which we describe in this subsection.

The main concern of the work by Chow and Liu [CL68] is to estimate an underlying n-dimensional joint probability distribution function, $P(\underline{X})$, $\underline{X} = [X_1, X_2,..., X_N]^T$ from a finite number of samples by using an approximation which is both easy to compute and represent. Since it requires enormous storage space to store estimates of the joint probability function $P(\underline{X})$ for all possible values of $\underline{X}$, we, first of all, observe that we will avoid maintaining all such estimates. Instead, Chow and Liu's algorithm constructs a dependence tree that best (i.e., the one which minimizes the information theoretic measure) approximates the underlying distribution $P(\underline{X})$ which utilizes only the information about the first and second order probabilities. The metric which they used to compare trees is the information theoretic metric I_f, which quantifies the similarity $D(P, P_a)$ between the true and the approximated distributions. An example of a dependence tree is given in Figure 2.1 below.

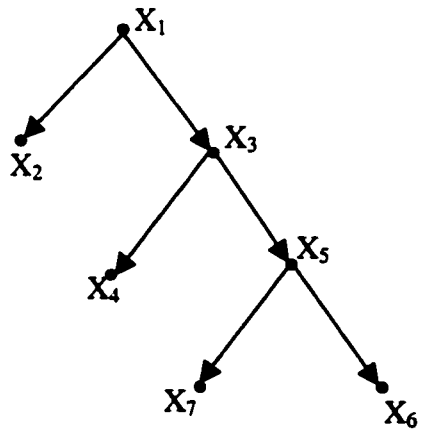


Figure 2.1: Example of a dependence tree in which X_1 is the root and $P(\underline{X} = [X_1, X_2,.., X_N]^T) = P(X_1).P(X_2 \mid X_1).P(X_3 \mid X_1).P(X_4 \mid X_3).P(X_5 \mid X_3).P(X_7 \mid X_5).P(X_6 \mid X_5)$.

To fully understand Chow and Liu's strategy we list the following important observations.

(i) Any tree-dependent distribution $P_t(\underline{X})$ can be written as a product of the first and second order probabilities as follows:

$P_t(\underline{X}) = \prod_{i=1}^{N} P(X_i / X_j(i))$, where the root X_1 has no parent and is characterized by the prior probability $P(X_1)$, and for all other nodes, $X_j(i)$ is the parent of X_i.

(ii) A graph G with N nodes can produce N^{N-2} spanning trees, where each tree is associated with a unique approximation of type:

$$P_t(\underline{X}) = \prod_{i=1}^{N} P(X_i / X_j(i))$$

The question which we now encounter is the following: which tree among these N^{N-2} trees is the best?

Chow and Liu [CL68] chose the Kullback-Leibler [KL51] cross-entropy measures to compare joint probability distributions. Thus they used $D(P, P_a)$ as a measure of the closeness of the approximation P_a to P, where,

$$D(P,P_a) = \sum_{\underline{X}} P(\underline{X}) \log \frac{P(\underline{X})}{P_a(\underline{X})}$$

Observe that

(i) $D(P,P_a) > 0$, if $P \neq P_a$

(ii) $D(P, P_a) = 0$ if and only if $P(\underline{X}) = P_a(\underline{X})$ for all $\underline{X}$.

They thus concluded that if we seek a good approximation P_a of P we should determine the tree structure which minimizes $D(P, P_a)$. Consider $D(P, P_a)$ for a given tree representation. In this case,

$$D(P, P_a) = D(P, P_t) = \sum_{\underline{X}} P(\underline{X}) \log \frac{P(\underline{X})}{P_a(\underline{X})} = \sum_{\underline{X}} P(\underline{X}) \log \frac{P(\underline{X})}{P_t(\underline{X})}, \qquad (2.1)$$

where $P_t(\underline{X}) = \prod_{i=1}^{N} P(X_i / X_j(i))$.

Expanding (2.1) we get

$$D(P, P_t) = - \sum_{\underline{X}} P(\underline{X}) \sum_{i=1}^{N} \log P_t(X_i / X_j(i)) + \sum_{\underline{X}} P(\underline{X}) \log P(\underline{X}) \qquad (2.2)$$

This equation can be written as:

$$D(P, P_t) = - \sum_{i=1}^{N} I_f(X_i, X_j(i)) + \sum_{i=1}^{N} H(X_i) - H(\underline{X}), \text{ where}$$

(i) $$H(X_i) = - \sum_{X_i} P_t(X_i) \log P_t(X_i),$$

(ii) $$H(\underline{X}) = - \sum_{\underline{X}} P(\underline{X}) \log P(\underline{X}) \text{ and}$$

(iii) $$I_f(X_i, X_j) = \sum_{X_i, X_j} P_t(X_i, X_j) \log \frac{P_t(X_i, X_j)}{P_t(X_i) P_t(X_j)}.$$

Because $H(\underline{X})$ and $H(X_i)$ are independent of the dependence tree and $D(P, P_t) \geq 0$, minimizing the difference between the two distributions $D(P, P_t)$ is equivalent to maximizing the total weight of the branches of the spanning tree $\sum_{i=1}^{N} I_f(X_i, X_j)$.

Consequently, Chow and Liu proved that the Maximum Weight Spanning Tree (MWST) obtained from the complete graph in which the edge weights are the quantities $I_f(X_i, X_j)$ yields the best tree which gives the best approximation to the underlying distribution. The algorithm to achieve this is formally given below.

Algorithm Chow/ Liu

Input: The probabilities $P(X_i, X_j)$ for all pairs of random variables X_i, X_j.

Output: The best dependence tree, which utilizes only these second order moments.

Method:

Begin

For (i =1 to N and j < i) **Do**

Compute $I_f(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) \log \frac{P(X_i, X_j)}{P(X_i) P_t(X_j)}$

EndFor

Repeat Until A Spanning Tree is completed

Include the branch with largest weight.

If cycle is obtained **Then**

Discard edge.

EndIf

EndRepeat

End Algorithm Chow/ Liu

The above algorithm utilizes the known joint distributions of X_i, X_j. If they are unknown, it is well known that we can estimate them from the samples. In such a case the tree obtained by the estimation is the Maximum Likelihood tree. This is proved formally by [VO92] and given in the next subsections.

The advantages of this algorithm are numerous. First of all, it uses only second-order probabilities, which are practical and economical to store. Also, as there will typically be enough samples matching any specified value, the conditional probability of X_i and X_j , $P(X_i \mid X_j)$ can be estimated reliably from the samples which may not necessarily be true if higher order moments are used[3]. The algorithm is also computationally efficient even though the approximating distributions utilize only second -order probabilities.

Experimental results which are reported in [VO92] demonstrate that when the underlying unknown distribution is derived from a dependence tree, the algorithm yields the underlying tree almost everywhere. Additionally, even if the data is not generated by a tree the algorithm constructs the tree that most closely approximates the underlying distribution from the perspective of the total (not just edge-wise) cross-entropy metric.

[3] The question of whether such estimates have meaningful significance remains open

2.4 Dependence Trees using Estimates of I_f (.,.) and the Chi-Squared Metric

The results of computing I_f and the subsequent tree have been applied to classification in which the aim was to classify a specific sample into one of a set of known classes (C_1, ..., C_h). If X represents the measurement made on a given sample, the problem involved determining the class C_i which maximizes the probability $P(C_i \mid \underline{X})$. Since this term can be computed only with the knowledge of the distribution $P(\underline{X} \mid C_i)$, Valiveti & Oommen [VO92] determined the class conditional distributions using the method discussed above.

As mentioned in the last subsection the first solution to the problem of approximating discrete distributions was presented by [CL68]. The authors of [CL68] utilized a relationship between the measure of closeness of two random variables and the measure of dependence between them. This measure of dependence between two variables (X_i , X_j) was the information theoretic measure (Kullback-Leibler cross-entropy measure)

$$I_f(X_i, X_j) = \sum_{X_i, X_j} P_t(X_i, X_j) \log \frac{P_t(X_i, X_j)}{P_t(X_i) P_t(X_j)},$$

where P_t is the probability distribution of the tree approximating the underlying distribution. As mentioned earlier, Chow & Liu used this measure to build the MWST dependence tree that best approximates the underlying probability. Indeed, when the probabilities are not known exactly but only the estimates are obtained from the samples, Chow & Liu stated that their algorithm finds the maximum likelihood estimate of the dependence tree. Thus the [CL68] algorithm determines the tree which maximizes the likelihood of generating the actual occurrence of samples from the N^{N-2} possible spanning trees.

This result is stated in Theorem 2.1 and illustrated in Figure 2.2. The theorem itself was alluded to by [CL68] but formally proved in detail by Valiveti & Oommen [VO92] by formally minimizing the likelihood function that would be obtained from the observed samples.

Theorem 2.1:

The dependence-tree produced by the [CL68] algorithm is the maximum likelihood estimate tree which can be obtained from the samples $\{ \underline{X}^1, \underline{X}^2, ... , \underline{X}^s \}$ themselves, where $\underline{X}^j$ represents an instance of the vector $\underline{X}$. □

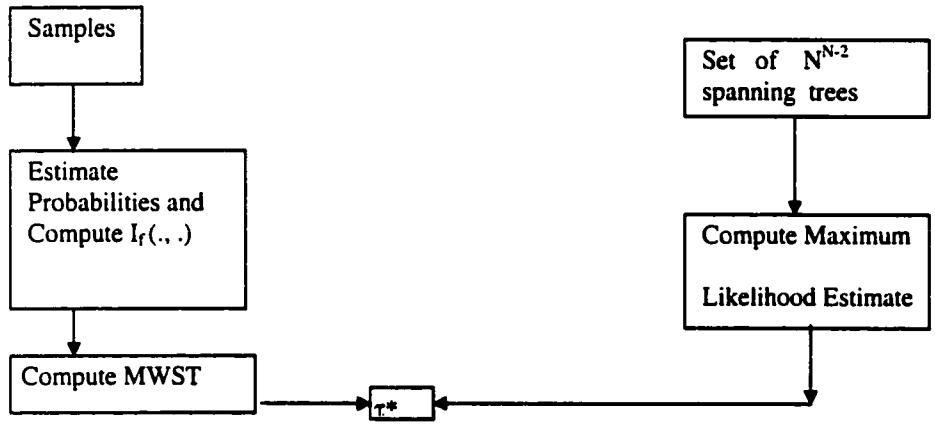


Figure 2.2: Equivalent procedures for finding the maximum likelihood tree from samples.

This figure shows the contribution of Theorem 2.1 and demonstrates that the two procedures, which can be used to find the maximum likelihood tree from the samples, are equivalent.

The main problem with the [CL68] strategy is that it searches the maximum weight dependence tree after performing computation of the information metrics $I_f(X_i, X_j)$ for all $\binom{N}{2}$ pairs of variables. This is quite cumbersome since it involves the computation of numerous logarithms for each of the $O(N^2)$ binary discrete variables. To circumvent these computations, Valiveti & Oommen in [VO92] presented a new chi-Squared metric I_χ which they used as a measure of dependence between pairs of discrete variables. The I_χ measure is defined as follows:

$$I_\chi(X_i, X_j) = \sum_{X_i X_j} \frac{(P(X_i, X_j) - P(X_i)P(X_j))^2}{P(X_i)P(X_j)}$$

Note again that just like the quantity $I_f(.,.)$,I_χ satisfies:

(i) $I_\chi(X_i, X_j) \geq 0$

(iii) $I_\chi(X_i, X_j) = 0$ if and only if $P(X_i, X_j) = P(X_i)\, P(X_j)$, in which case X_i and X_j are statistically independent.

It is clear that the computation of I_χ for each edge weight is faster than the corresponding computation using I_f. Having obtained I_χ for all the $\binom{N}{2}$ edges, [VO92] determined the best dependence tree using these weights by again invoking the MWST algorithm. They proved that their algorithm yields the maximum likelihood estimate if the weights are the I_χ, and also showed that I_χ increases and decreases monotonically with I_f for binary random variables.

This new metric I_χ is theoretically non-optimal (see [VO92] for the details of the non-optimality) when used for computing general probability distributions represented by trees. However, the I_χ metric satisfies the following properties:

(i) In the case when the random variables X_i, X_j are binary valued, the metric $I_\chi(X_i, X_j)$ increases or decreases monotonically with $I_f(X_i, X_j)$.

(ii) For a restricted class of probability distributions (see [VO92] for details) which represents a specific type of dependence between the individual random variables, the quantities $I_\chi(X_i, X_j)$ and $I_f(X_i, X_j)$ are equivalent. Thus within this family, the I_χ metric indeed yields the maximum likelihood estimate of the underlying tree, even though the computation is achieved without computing the matrix $I_f(.,.)$.

Valiveti & Oommen [VO92] have done numerous experiments to show that the I_χ metric is very accurate. The results demonstrate that if the data is generated by an underlying distribution that can be represented as a tree, both metrics I_χ and I_f succeed in finding the *exact* same tree after a small, reasonable number of (testing) samples. Also even if the underlying dependence is not tree dependence, both estimates yield exactly the same

"best dependence" tree. The main advantage with the new metric is, of course, that the computations are faster than with I_f. Valiveti & Oommen [VO92] also gave a detailed and interesting discussion on the accuracy of this new metric. Using these principles they developed a classification system whose accuracy was quite high. The details of the experimental setup and the results obtained are given in [VO92].

2.5 Dependence Trees for Continuous Random Vectors

In the previous sections we considered the problem of approximating an unknown underlying n-dimensional discrete joint probability distribution function, $P(\underline{X})$ where $\underline{X} = [X_1, X_2, ..., X_N]^T$ was a binary vector of length N and the samples are drawn independently based on $P(\underline{X})$. The dependence tree that best approximates the "real" distribution was proven to be the maximum weight spanning tree of a graph G having X1, X_2,..., X_N as nodes, with edge weights defined as follows:

$$I_f(X_i, X_j) = \sum_{X_i, X_j} P_t(X_i, X_j) \log \frac{P_t(X_i, X_j)}{P_t(X_i) P_t(X_j)},$$

where I_f is the information theoretic metric, and P_t is the tree distribution approximating the underlying distribution.

Chow *et al.*, in [CWS79] also studied the case when the vector $\underline{X} = [X_1, X_2,..., X_N]^T$ was composed of continuous normal random variables $\{X_i\}$ and $P(\underline{X})$ was the joint density function of the random vector $\underline{X}$. Again the problem was one of approximating the unknown probability distribution function $P(\underline{X})$ by one using a tree-based dependence. Chow *et al.*, utilized the information theoretic metric I_f and designed an algorithm identical to the one described in Section 2.3. Later Valiveti & Oommen [VO93] derived analogous results to [CWS79] except that as in [VO93] they used the I_χ metric instead of I_f. The power of their result of approximating a distribution by a tree-based distribution is that the computation can be done without inverting the covariance matrix Σ. They showed that just as in the case of the EMIM metric, the Chi-Squared metric for continuous normal vectors yields the *optimal* tree. They also proved that the

tree obtained by using the estimate $\hat{\Sigma}$, of the covariance matrix Σ (when it is not known exactly) is the Maximum Likelihood Estimate (MLE) of the dependence tree, and that this estimate converges to the true underlying tree as the number of samples increases. The details of these results are explained below.

A joint distribution is said to be of tree dependence if $P(\underline{X})$ can be written as follows: $P(\underline{X}) = \prod_{i=1}^{N} P(X_i \mid X_j(i))$.

In the above we observe that since we are dealing with continuous random variables, we utilize the probability density functions instead of the probability masses.

To determine the best approximating density function, Chow *et al.*, in [CWS79] used the metric for closeness between the "real" density function $P(\underline{X})$ and the approximating density $P_a(\underline{X})$ as:

$$I(P, P_a) = \int P(\underline{X}) \log \frac{P(\underline{X})}{P_a(\underline{X})} d\underline{X}.$$

It can be seen that as in the discrete case, I is a metric, and so it obeys:

(i) $I(P, P_a) \geq 0$, and

(ii) $I(P, P_a) = 0$ if $P(\underline{X}) = P_a(\underline{X})$.

If τ is the dependence tree approximating the underlying "real" unknown distribution,

$$I(P, P_\tau) = A - \sum_{(i,j) \in \tau} I_f(X_i, X_j),$$

where A is a constant independent of the approximation, and hence

$$I_f(X_i, X_j) = E\left[\log \frac{P(X_i, X_j)}{P(X_i)P(X_j)}\right].$$

As explained in Section 2.2, in the discrete case, the evaluation of the information metric between pairs of variables only depends on the actual discrete probabilities of the joint events. In the continuous case, since we deal with probability densities, there is no straightforward method to numerically compute the information metric $I_f(X_i, X_j)$ if the analytic form of the density function is not known. Furthermore even if it is known, the

quantity $I_f(X_i, X_j)$ may not be computable if the integrals cannot be evaluated in their closed forms (See pp.59 of [Val92]). As a consequence of this, the problem of finding the dependence tree of an arbitrary random vector $\underline{X} = [X_1, X_2, ..., X_N]^T$ where the variables are continuous was studied by Chow [Cho68] only for the case when these variables are normally distributed.

In this case, if $\underline{X} = [X_1, X_2, ..., X_N]^T$ is jointly normal with mean vector $\underline{\mu} = [\mu_1, \mu_2, ..., \mu_N]^T$ and the covariance matrix $\Sigma = [\sigma_{ij}]$, since each X_i is $N(\mu_i, \sigma_i)$, its density function is:

$$P(X_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\{-\frac{1}{2}\left(\frac{X_i - \mu_i}{\sigma_i}\right)^2\},$$

also the density function of $\underline{X}$ is:

$$P(\underline{X}) = (2\pi)^{-N/2} |\Sigma|^{-0.5} \exp\{-\frac{1}{2}(\underline{X} - \underline{\mu})^T \Sigma^{-1}(\underline{X} - \underline{\mu})\}.$$

Finally, it can be shown that for all jointly normal pairs of random variables X_i and X_j, the information metric $I_f(X_i, X_j)$ is given by:

$$I_f(X_i, X_j) = E\left[\log\frac{P(X_i, X_j)}{P(X_i)P(X_j)}\right] = -\frac{1}{2}\log(1 - \rho_{ij}^2),$$

where ρ_{ij} is the correlation coefficient.

Using these relationships, the authors of [VO93] report three algorithms to find the best dependence tree, which approximates the underlying distribution. The first algorithm due to Chow *et al.*, in [CWS79] assumes that the random vector $\underline{X} = [X_1, X_2, ..., X_N]^T$ is jointly normal and the true values of the mean vector $\underline{\mu} = [\mu_1, \mu_2, ..., \mu_N]^T$ and the covariance matrix $\Sigma = [\sigma_{ij}]$ are known. As in [CWS79], if the information metric I_f is used to compute the edge weights of the tree then a MWST yields the optimal estimate of the best tree. The second algorithm makes the same assumption about the mean vector and the covariance matrix, but uses a more computationally effective metric, the chi-Squared metric for determining the dependence tree described in section 2.3. Valiveti & Oommen [VO93] show that the Chi-Squared metric *always* yields the optimal tree the one exactly equal to the tree obtained by the I_f for the normal case. The last

algorithm deals with the case when the true values of the mean vector and the covariance matrix are not available, but only a finite number of samples are available to the algorithm which estimates the dependence tree. In that case the maximum likelihood estimate of the covariance matrix is first obtained from the samples, and then the algorithm uses *it* to yield the maximum likelihood estimate of the dependence tree. Convergence results of the latter case are also included in [VO93]. These three algorithms are formally given below.

Algorithm Normal-Chow
Input: The parameters $\underline{\mu}$ and Σ for the normal distribution obeyed by $\underline{X}$
Output: The best dependence tree obtained using the I_f measure: τ^*.
Method
Begin
Compute R := $[\rho_{ij}]$, the matrix of correlation coefficients from Σ.
For (i =1 to N and j < i) **Do**
Compute the information measure $I_f(X_i, X_j) = -\frac{1}{2}\log(1 - \rho_{ij})$
EndFor
τ^* = MWST from the set of nodes V and edge weights $\{I_f(X_i, X_j)\}$.
End Algorithm Normal-Chow

Consider the case when the Chi-Squared metric is used to quantify the dependence between the variables. I_χ is obtained by computing the χ^2 distance between pairs of random variables. For continuous random variables the expression for I_χ is:

$$I_\chi(X_i, X_j) = \iint \frac{\{P(X_i, X_j) - P(X_i)P(X_j)\}^2}{P(X_i)P(X_j)} dX_i dX_j$$

As in the case of I_f, Valiveti & Oommen [VO93] show that if $\underline{X}$ has a Gaussian distribution $N(\underline{\mu}, \Sigma)$, a closed form expression for I_χ can be given as:

$$I_\chi(X_i, X_j) = \frac{\rho_{ij}^2}{1 - \rho_{ij}^2}$$

Using these, the algorithm for τ_χ can be written as follows:

Algorithm Normal Valiveti/Oommen
Input: The parameters $\underline{\mu}$ and Σ for the normal distribution obeyed by $\underline{X}$.
Output: The best dependence tree obtained using the I_χ measure: τ_χ.
Method
Begin
Compute R := $[\rho_{ij}]$, the matrix of correlation coefficients from Σ.
For (i =1 to N and j < i) **Do**

$$I_\chi(X_i, X_j) = -\frac{1}{2}\log(1 - \rho_{ij})$$

EndFor
τ_χ = MWST from the set of nodes V and the edge weights $\{I_\chi(X_i, X_j\}$.
End Algorithm Normal Valiveti/Oommen

The similarity between the two algorithms essentially follows from the fact that second order properties of the normal distributions completely describe the underlying tree.

As mentioned above, the authors of [VO92] showed that for any two components X_i and X_j of the normal random vector $\underline{X}$, $I_\chi(X_i, X_j)$ is a monotonic function of $I_f(X_i, X_j)$. Thus, if $\underline{X}$ is a normally distributed random vector, then τ_χ obtained by the algorithm using I_χ measure is either identical to, or is an equally good (based on I_f) approximation as the tree τ^* produced by the algorithm using the I_f metric.

The previous algorithms used the assumptions that the "true" parameters of the normal distribution of the random vector $\underline{X}$ are known. But the more realistic scenario is when we just have samples of the vector $\underline{X}$. In such a case we need to first compute the estimates of the parameters from the samples and thereafter compute the estimate of the tree itself. The algorithm below derives first the MLE $\hat{\Sigma}$ of covariance matrix. It thereafter uses these estimates to obtain the dependence tree either by the algorithm using the information metric or the chi-Squared metric.

Algorithm Dependence-Tree-from-Samples
Input: s statistically independent samples $\underline{X}^1, \underline{X}^2, .., \underline{X}^s$.
Output: An estimate of the dependence tree, $\hat{\tau}$, as per the chi-squared metric.
Method
Begin

Compute the maximum likelihood estimates of the parameters of the distribution of $\underline{X}$.

$$\hat{\underline{\mu}} := \frac{1}{s}\sum_{k=1}^{s} \underline{X}^k \quad \text{and}$$

$$\hat{\Sigma} = \frac{1}{s}\sum (\underline{X}^k - \hat{\underline{\mu}})(\underline{X}^k - \hat{\underline{\mu}})^T .$$

Call Normal Valiveti/Oommen (input: $\hat{\underline{\mu}}$, $\hat{\Sigma}$ and output: $\hat{\tau}$)

End Algorithm Dependence-Tree-from-Samples

The interesting property of the above algorithm is that the estimate obtained by invoking the MWST procedure using the parameter estimates is exactly the maximum likelihood of the true underlying tree. This is proved by [VO93] as stated in the theorem below.

Theorem 2.2:

The algorithm Dependence Tree which utilizes the estimates for $\underline{\mu}$ and Σ produces the Maximum Likelihood Estimate of the dependence tree. □

Experimental results again demonstrate that when the underlying dependence is of the tree type, the estimate of the dependence tree converges to the underlying unknown tree as the number of samples increases. However, if the underlying dependence is not of the tree type, the estimated tree is the best tree which models the covariance Σ. In the case of normal vectors both the Chi-Squared metric and the information theoretic metric can be shown to be exactly equivalent conceptually and computationally.

2.6 Recovering Causal Polytrees

A polytree (singly connected network) is a belief network that contains at most one undirected path (i.e., a path that ignores the directions of arcs) between any two nodes in

the network. An example of such a polytree is given in Figure 2.3. Figure 2.4 shows a non-polytree graph.

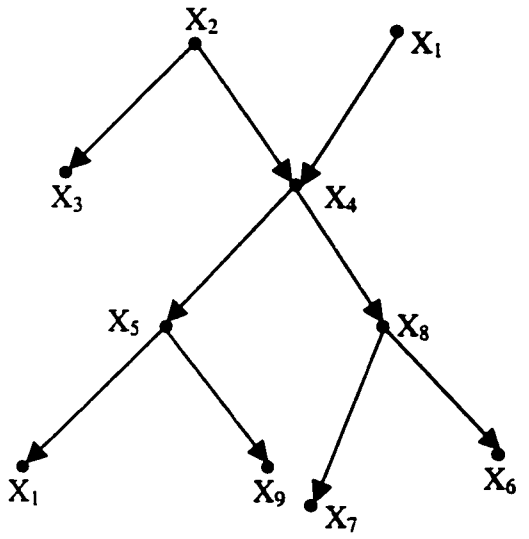


Figure 2.3: A polytree

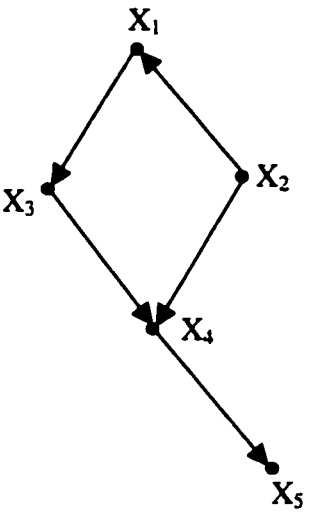


Figure 2.4: A non polytree

From the point of view of stochastic distributions, a distribution $P(\underline{X})$ that can be represented by a polytree is written as follows:

$$P(\underline{X}) = \prod_{i=1}^{N} P(X_i \mid X_{j1}(i), X_{j2}(i),..., X_{jm}(i)),$$

where $X_{j1}(i)$, $X_{j2}(i)$,....,$X_{jm}(i)$ are parents of X_i and are statistically independent. Consequently,

$$P(X_{j1}(i), X_{j2}(i),....,X_{jm}(i)) = \prod_{j=1}^{m} P(X_{ij}(i)) \text{ for all i.}$$

Definition 2.7 [Pea88]: A distribution $P(\underline{X})$ is said to be **nondegenerate** if it has a connected, **perfect map**, i.e., if there exists a directed acyclic graph that displays all the dependencies and independencies embedded in P. More explicitly, for every three disjoint subsets X, Y, and Z, such a distribution obeys:

$$I(X, Z, Y) \Leftrightarrow < X \mid Z \mid Y >.$$

Rebane and Pearl [RP87] developed an algorithm that recovers polytrees for random variables whose stochastic properties can be described in the form of a discrete joint probability density function. To achieve this they first invoked a "Chow type" [CL68] MWST to construct the skeleton of the polytree, which yielded the undirected structure of tree. We shall now explain how the orientation of the arcs is subsequently computed.

Given a tree-type dependence of undirected edges, Rebane & Pearl [RP87] determine the orientation of the arcs by using a dependency structure of type

T1: $X \rightarrow Z \leftarrow Y$

Where X and Y are marginally independent variables. To test this kind of dependency, an independency test is used to determine if a variable has multiple parents. In fact, every two node neighbors X and Y of a node Z are considered and tested to see if they are marginally independent, and hence to decide if node Z has parents X and Y.

Rebane and Pearl also state that a partially directed triplet $X \rightarrow Z —$ Y can be oriented by testing for the independence of X and Y. If X and Y are independent, Y is a parent of Z, otherwise Y is a child of Z.

If we are given a distribution $P(\underline{X})$ of N discrete value variables and we know that $P(\underline{X})$ can be represented by a polytree and that $P(\underline{X})$ is a non-degenerate distribution, Rebane and Pearl [RP87] guarantee that the polytree algorithm sketched above yields the skeleton (tree) of the polytree. They also guarantee that the directions of the arcs can be determined to the maximum extent permitted by $P(\underline{X})$ by repeatedly invoking the independence tests described above. Indeed, the non-degeneracy restriction guarantees that the MWST yields the skeleton of the polytree, and hence the polytree itself can be recovered. The following two theorems constitute the theoretical basis of their algorithm.

Theorem 2.3 [Pea88]:

If a non-degenerate $P(\underline{X})$ is representable by a polytree, the MWST algorithm can be used to unambiguously recover its skeleton. □

The condition for recovering the directionality of the edges is given by the following theorem.

Theorem 2.4 [Pea88]:

The directionality of a branch can be recovered if and only if it is contained within some causal basin of the polytree. □

We would like to informally state that in a DAG, a set of connected edges constitutes a causal basin[4] if the orientation of the edges is in the same direction in terms of causal knowledge. The example below shows a causal basin. The formal definitions are found in Chapter 3.

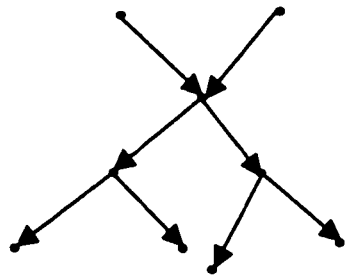


Figure 2.5: An example of a causal basin

Theorem 2.4 states that once we have obtained the skeleton of the polytree, every node with multiple neighbors must be considered. The independence test can then be used to verify if two neighbors are marginally independent. When the first pair of parents are found, the succeeding generations of descendants are resolved using the partially directed triplet and the process continues in the direction of the causal basin (flow). This leads us to the following scheme due to [RP87].

[4] This description of a causal basin is identical to what has been specified by Pearl [Pea88]. In Chapter 3 we shall describe how it can be computationally obtained – depending on the starting vertex.

Algorithm Polytree-Rebane/Pearl
Input: A tree and independency tests for the nodes
Output: A polytree
Method
Begin

Generate a maximum weight undirected spanning tree (a skeleton), using the MWST algorithm.
Repeat Until no more further directionality can be discovered
(i) Search the internal nodes of the skeleton, beginning with the outermost layer and working inward, until a multi- parent node is found using the T1 test.
(ii) When a multi- parent node 'j' is found, determine the directionality of all of its branches using test T1.
For each node having at least one incoming arrow **Do**
resolve the directionalities of all of its remaining adjacent branches using test T1.
EndFor
End Repeat
If there remain undirected branches **Then**
(i) Label them "undirected"
(ii) Supply external semantics using which they can be directed.
EndIf

End Algorithm Polytree-Rebane/Pearl

As mentioned earlier, the structure of the polytree can be identified by second-order statistics using the MWST algorithm. This algorithm thus inherits many of the advantages of the MWST algorithm. Furthermore, a polytree structure is richer than a tree structure since $P(\underline{X})$ uses higher order probabilities.

In determining the orientation of arcs, the polytree algorithm assumes the availability of a conditional independence test which is a test that determines categorically whether the predicate I(X, ∅, Y) is true or false for variables X and Y. If all such tests fail the algorithm itself fails and consequently no polytree can be derived. As we shall see later, in practice it is almost impossible to obtain fully independent random variables with real life data and so the problem of studying real-life data remains open. This will be one of the topics, which we shall study later.

A second drawback of the Rebane/Pearl algorithm is that in degenerate cases, the algorithm may not return the structure of the underlying belief network. Finally, on encountering a probability distribution P that cannot be represented by a polytree, the

algorithm yielded by Rebane/Pearl is not guaranteed to produce a polytree that most closely approximates P. The main drawback of the Rebane/Pearl algorithm is that the question of using it to tackle real-life data is still unsolved, which is probably why the paper [RP87] does not contain any experimental results.

2.7 Learning Causal Trees from Independence Information

One of the restrictions of the polytree recovery algorithm is that the distribution P is assumed to be non-degenerate. As stated before, a distribution $P(\underline{X})$ is said to be **non-degenerate** if there exists a directed acyclic graph that is a **perfect map** of the dependency model i.e., it displays all the dependencies and independencies of the model. In other words as stated in [GPP90], the dependency model satisfies the following five independent axioms:

A_1 **Symmetry**:

$I(X, Z, Y) \Leftrightarrow I(Y, Z, X)$

A_2 **Decomposition:**

$I(X, Z, Y \cup W) \Rightarrow I(X, Z, Y)$ & $I(X, Z, W)$

A_3 **Intersection:**

$I(X, Z \cup W, Y)$ & $I(X, Z \cup Y, W) \Rightarrow I(X, Z, Y \cup W)$

A_4 **Strong union:**

$I(X, Z, Y) \Rightarrow I(X, Z \cup W, Y)$

A_5 **Transitivity:**

$I(X, Z, Y) \Rightarrow I(X, Z, \gamma)$ or $I(\gamma, Z, Y)$ where γ is a single variable and the others disjoint sets of variables. □

Geiger *et al.* [GPP90] developed an algorithm to recover polytrees by using less restrictive assumptions about the distribution. Their algorithm assumes first, that a distribution P is given and that we know that P is represented by a singly-connected DAG D (or a polytree) whose structure is unknown. Secondly, their algorithm assumes that the distribution P satisfy axioms A_6, A_7 and A_8 below:

A_6 **Intersection:**

$I(X, X \cup Y, W) \& I(X, Z \cup W, Y) \Rightarrow I(X, Z, Y \cup W)$

A_7 **Composition:**

$I(X, Z, Y) \& I(X, Z, W) \Rightarrow I(X, Z, Y \cup W)$

A_8 **Marginal Weak Transitivity:**

$I(X, \varnothing, Y) \& I(X, c, W) \Rightarrow I(X, \varnothing, c)$ or $I(c, \varnothing, Y)$. □

Assuming these properties for the distribution P, the algorithm ensures that the DAG is recoverable in polynomial time. This algorithm is considered a generalization of the [RP87] algorithm as it makes fewer restrictions on the distribution. The authors show that any assumption of a (multivariate) normal distribution is sufficient for a complete recovery of singly connected DAGs since properties A_6, A_7 and A_8 are satisfied by normal distributions.

The [GPP90] recovery algorithm uses queries or assertions of the form I(X, Z, Y) as input which means that there is no computation of independence tests to state whether I(X, Z, Y) is true or false, and therefore it does not directly use any database of cases or samples from real life data. It starts with a complete graph and first removes all arcs X — Y for which the conditional independence I(X, Z, Y) holds with $Z \in U \setminus \{X, Y\}$. In fact, if Z separates X from Y then X — Y should not exist. Second, the algorithm removes all the arcs X — Y for which X and Y are marginally independent, i.e., $I(X, \varnothing, Y)$ is true. Then it orients the resulting graph using the same partially undirected triplets that we have already seen in the [RP87] algorithm.

Algorithm Recovery/Geiger

Input: Independence assertions of the form I(X, Z, Y) drawn from a pseudo-normal dependency model M represented by the triplets (X, Z, Y) and denoted by (X, Z, Y) in the algorithm below.

Output: A polytree I-map of M if it exists, or acknowledgment that no such I-map exists.

Method

Begin

Start with a complete graph G

For every edge X — Y for which (X, U \{X, Y}, Y) is in M **Do**

Remove edge X — Y from G and obtain the unoriented network G_0.

EndFor

For every edge X — Y for which (X, ∅ ,Y) is in G_0 **Do**

Remove edge X — Y from G_0 and obtain the unoriented network G_R.

EndFor

If (the resulting graph G_R has a cycle) **Then**

Answer "No". Exit.

EndIf

For every edge X — Y in G_R **Do**

If (Y has a neighboring Z such that (X, ∅, Z) ∈ M
and X — Z is in G_0) **Then**
Orient edge X → Y

EndIf

EndFor

If (the resulting orientations is not feasible) **Then**

Answer "No". Exit.

EndIf

If (the resulting polytree is not an I-map) **Then**

Answer "No".

Else

This polytree is a minimal-edge I-map of M.

EndIf

End Algorithm Recovery/Geiger

The algorithm Recovery/Geiger runs in polynomial time in the number of independence assertions and it is not just restricted to distributions that can be represented by polytrees. Furthermore, there is no inference or computation of independency tests to state whether I(X, Z, Y) is true or false. The algorithm constructs the structure from the insight of human experts instead of a database of cases. It relies on human expert to define the graphical structure.

One question pertinent to the algorithm is the following: Does the input provided possess all the required assertions for the accurate convergence of the algorithm or do we implicitly consider the well known Closed-World Assumption (CWA). This in turn means that if we don't have any information about I(X, Z, Y) we will consider it to be false; Indeed everything that is not known to be "True" will be considered to be "False". The implications of this assumption are yet unknown. One of the disadvantages of the algorithm is that it starts with a complete graph. This is space consuming, and is not very practical from an implementation point of view. Unfortunately, the authors report no experimental results in the paper [GPP90]. Furthermore their algorithm has not been applied to any application (whether it be with synthetic or real-life data).

Another drawback of the algorithm is the following: it uses higher order probabilities than the Pearl algorithm and the question one may ask is how reliable the estimates of these probabilities will be when we approximate them from samples.

2.8 Bayesian Induction of Probabilistic Networks

In [CH92], the authors use a different approach than all the previous work and this incorporates the principles of Bayesian learning. This philosophy examines the space of possible networks to search for network structure hypothesis possessing a high relative *a posteriori* probability. The key contribution of this work was to derive a formula for the computation of this probability.

As mentioned earlier, a Bayesian belief network structure B_s is a DAG in which nodes represent domain variables and arcs between nodes represent probabilistic dependencies. The network structure, B_s, is augmented by conditional probabilities, B_p, on the nodes to form the complete Bayesian belief network $B = (B_s, B_p)$.

The authors of [CH92] have described a Bayesian learning algorithm called K2 for constructing or learning the qualitative and quantitative dependency relationships among a set of discrete variables from empirical observations. Given a database D of cases representing a set of variables $\mathbf{Z}$, the algorithm searches for the most probable belief network structure, i.e., the structure B_s that maximizes the *a posteriori* probability $P(B_s|D)$ and derives a set of conditional probabilities B_p on every node of the structure,

thus building a Bayesian network $B = (B_s, B_p)$. The problem encountered is to find a method for determining the B_s that maximizes $P(B_s \mid D)$ using a scheme that is more efficient than that of exhaustive enumeration. Following the Bayesian law the *a posteriori* probability satisfies:

$$P(B_s \mid D) = \frac{P(Bs\ ,\ D)}{P(D)}\ , \text{and}$$

$$P(D) = \sum_{Bs \in Q} P(Bs\ , D) \quad \text{where,}$$

$$P(B_s \mid D) = \frac{P(Bs\ , D)}{\sum_{Bs \in Q} P(Bs\ , D)} \quad \text{where Q is the set of all structures containing Z.}$$

Consequently, we see that $P(B_s\ ,\ D) \approx P(B_s \mid D)$, and therefore maximizing $P(B_s \mid D)$ is equivalent to finding the B_s that maximizes $P(B_s\ ,\ D)$.

[CH92] have investigated the computation of the expression $P(B_s\ ,\ D)$. They have discussed how to find the most probable belief network structure and presented an algorithm, which derives the conditional probabilities on the nodes. The algorithm requires the user to provide an ordering of the variables. It evaluates the posterior probability of adding each possible arc to an initially empty graph and makes the highest-ranking addition to the graph. This greedy search is continued until no improvement is obtained for the posterior probability.

They have also introduced techniques for handling missing data and hidden variables and finally introduce applicable techniques for probabilistic inference. The details of the computation follow in the next subsection.

2.8.1 Computing $P(B_s, D)$ for BBNs

Let D be a database of cases, Z be the set of variables represented by D and B_s be belief network structure containing the variables in Z. To compute $P(B_s, D)$, the following four assumptions are used ($As_1 - As_4$):

As_1: The variables are discrete.

As_2: Cases occur independently given a belief network model.

As_3 There are no cases that have variables with missing values.

As_4: The density function $f(B_p \mid B_s)$ is uniform. Consequently,

$$P(B_s, D) = \int_{B_p} P(D|B_s, B_p) f(B_p|B_s) P(B_s) dB_p$$

Where, f( ) is the conditional probability density function over B_p given B_s, and $P(B_s)$ is the prior probability.

In the absence of other information we can assume that the prior probability is the same for all the structures setting $P(B_s)$ to be a constant 'c'. $P(D|B_s, B_p)$ is the probability of observing the data in D given a belief network with structure B_s and with probabilities B_p. Thus,

$$P(B_s, D) = \int_{B_p} \left[\prod_{h=1}^{m} P(C_h|B_s, B_p) \right] f(B_p|B_s) P(B_s) dB_p$$

Where m is the number of cases and C_h is the h^{th} case in D.

The authors in [CH92] have proved the following fundamental results. Let Z be a set of N discrete variables where a variable X_i in Z has r_i possible value assignments: (V_{i1}, V_{i2},...,V_{iri}). Let D be a database of N cases where each case contains a value assignment for each variable in Z. Each variable X_i in B_s has a set of parents, which we represent with a list of variables $\prod_i$. Let w_{ij} denote the jth unique instantiation of $\prod_i$ relative to D. Suppose there are q_i such unique instantiations of $\prod_i$. We define N_{ijk} to be the number of cases in D in which the variable X_i has the value V_{ik} and $\prod_i$ is instantiated as W_{ij}. Let $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ Then, if the *a priori* probability of all the structures is equal to c, the probability $P(B_s, D)$ is:

$$P(B_s, D) = c \prod_{i=1}^{N} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \qquad (2.3)$$

We now consider the computation of most probable network structure. For a given database D we can maximize $P(B_s, D)$ by applying (2.3) exhaustively for every possible B_s but this is not feasible since the number of structures is very large. To maximize (2.3),

the authors of [CH92] have concentrated on finding a set of parents $\prod_i$ of a variable that maximizes the product $\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}!$. Subsequently they assign:

$$\max_{B_S}[P(B_s,D)] = c\prod_{i=1}^{N}\max_{\pi_i}[\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}!], \text{ and}$$

$$\max_{B_S}[P(B_s,D)] = c\prod_{i=1}^{N}\max_{\pi_i}[P(\pi_i \rightarrow x_i)\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}!].$$

Clearly the exhaustive search of the space is infeasible. Instead Cooper and Herskovits[CH92] have devised a polynomial time heuristic search method for maximizing $P(B_S, D)$ which assumes an ordering on the variables. For each node, it starts with an empty set of parents and then incrementally adds a parent that maximally increases the following function:

$$g(i, \pi_i) = \prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}! \qquad (2.4)$$

This "greedy" procedure stops adding parents to the node when there is no increase in g(node, parents(node)). The algorithm due to Cooper *et al.* [CH92] repeats the same process to all the nodes. The algorithm is formally given below.

Algorithm Cooper/Herskovits

Input: A set of N nodes, an ordering on the nodes, an upper bound u on the number of parents a node may have, and a database D containing m cases or observations.
Pred(X_i) returns the set of nodes that precede X_i in the node ordering.

Output: For each node, a printout of its parents.

Method

Begin

For i =1 **to** N **Do**

$\prod_i$ = empty

P_{old} = g(i, $\prod_i$)

cont = true

While cont = true and |$\prod_i$ | < u **Do**

Let Z be the node in (Pred(X_i) - $\prod_i$) that maximizes g(i, $\prod_i$ U{Z}) where g(.,.) is given by (2) above

P_{new} = g(i, $\prod_i$ U{Z})

If P_{new} > P_{old} **Then**

P_{old} = P_{new}

$\prod_i$ = $\prod_i$ U{z}

Else

cont = false

EndIf

EndWhile

Write ($\prod_i$ 'are the parents of X_i ', X_i)

EndFor

End Algorithm Cooper/Herskovits

We now consider how we can derive the probabilities on every node of the structure. Indeed, the conditional probabilities on the nodes of the belief network structure are given by using the following theorem proved in the appendix of [CH92].

Theorem 2.9 [CH92]:

Let θ_{ijk} = P(X_i =v_{ik} | parents(X_i) = w_{ij}) where θ_{ijk} is the conditional probability that X_i has value v_{ik} given that parents(X_i) have value parents(X_i) = w_{ij}. Let ξ denote the set of

assumptions (A_1 – A_4). If $E[\theta_{ijk} \mid D, B_s, \xi]$ is the expected value of θ_{ijk} given the database D, the belief network structure B_s and the assumptions ξ,

$$E[\theta_{ijk} \mid D, B_s, \xi] = \frac{N_{ijk}+1}{N_{ij}+r_i} \qquad \square$$

By computing the set of values $E[\theta_{ijk} \mid D, B_s, \xi]$ of all the nodes of the structure already obtained from the previous algorithm, we get the conditional probabilities of the Bayesian network which will achieve the learning of BBNs.

From the point of view of the disadvantages of the scheme the algorithm developed by the [CH92] is not optimal, and thus the Bayesian belief network B obtained is not the best graph that fits the data. The construction algorithm is not space efficient. Finally the algorithm is only suitable for discrete variables, and so does not handle the scenario when we deal with continuous variables.

This paper is considered as a seminal paper, which gave rise to a stream of research within a Bayesian framework ([Bun94], [RS97]). Also some related works to the Bayesian learning of BBNs are given in [HGC95] and [FG96]. Heckerman *et al.*, [HGC95] derived a modification to the K2 algorithm for computing the posterior probability and a local search algorithm that uses the improvement. Their method requires a prior probability distribution in the form of a prior network.

Friedman and Goldszmidt [FG96] developed a network learning algorithm, called tree-augmented naïve Bayes (TAN) which starts with a naïve Bayes network and considers adding arcs to improve the posterior probability of the network.

2.9 Summary

The main aim of this chapter is to review some existing methods to learn efficient structures, which allow the inference of causality. We have covered two main approaches. The first approach is based on probabilistic- graph models (section 2.3 through 2.6). The second approach is the Bayesian learning method (section 2.7). The probabilistic methods for learning BBNs focus on finding the most likely structure, implicitly assuming that there is a true underlying structure. It uses the various conditional independence assertions to determine that structure.

Along the Bayesian learning approach, there are two main tasks, which are involved in a procedure, which learns the BBN from cases. The first task attempts the induction of the graphical model which best fits the database, and the second task deals with the inference of the conditional probabilities on the nodes. The first task exploits heuristics to reduce the search space of all possible graphical models, and uses a score metric to drive the search process and assess the goodness of fit of a structure.

Chapter 3

Learning Polytrees from Probability Distributions

3.1 Introduction

In the previous chapter, we presented a review of existing algorithms for building belief networks from distributions. We studied Bayesian techniques and non-Bayesian techniques. This chapter deals with the problem of automatically building a BBN with the assumption that the observations have been presented to the system in terms of joint probability distributions. Thus we assume, in this chapter, that the joint dependence relations represented by these observations is available. Pearl [Pea88] discussed this process using a two-phase dependence learning scheme: The first phase learns the topology of the dependency model for a probability distribution, and the second learns the conditional probabilities for a given network structure.

In this chapter, we focus on learning causal structures, and, in particular, we consider learning polytrees. Indeed, more specifically, our aim is to find causal polytree structures that fit our data presented in terms of joint probability distributions. Since causality plays an important role in human understanding, causal knowledge is a major part of any intelligent system.

We intend to consider whether we can *approximate* every belief network as a causal structure. In section 2.3, we presented the algorithm developed by Chow *et al.* [CL68], which returns an undirected dependency tree. The authors of [CL68] showed that an arbitrary joint distribution could be optimally approximated by a tree-dependent distribution. They considered the approximation of an nth order distribution with N-1 second order distributions using the Expected Mutual Information Metric (EMIM) Kullback-Leibler cross-entropy measure. As explained in section 2.1, in their algorithm they compute $O(N^2)$ quantities which correspond to the information metric between any two variables. To obtain the best tree, a greedy MWST algorithm was invoked which, at

every stage, retained the edge with the highest measure until an N-1 branched tree was obtained.

In Section 2.5, we explained how Rebane and Pearl [RP87] extended the tree approximation algorithm to causal polytrees. Their algorithm invoked the Chow/Liu algorithm to first recover the skeleton of the polytree and thus takes advantage of the second order probabilities. They subsequently invoked conditional independence tests to orient the arcs of the tree.

The reader must observe that polytrees represent much richer dependency models than undirected trees, because their joint probability density functions can be products of higher order distributions. However, the problem itself is shown to be a much harder problem than that of finding the best tree. First of all, the algorithm is not guaranteed to find a polytree structure if the underlying distribution is degenerate and not of a polytree type distribution i.e., if the distributions do not fit into a polytree representation. Secondly, the algorithm relies on the repeated use of the independence test:

$$I_f(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) \log \frac{P(X_i, X_j)}{P(X_i)P(X_j)}$$

that determines categorically whether two random variables X_i and X_j are statistically independent i.e., $I_f(X_i, X_j) = 0$. As we shall see in a later chapter, even if the random variables are statistically independent, the experimental evaluation of them may never be close enough to zero with real-life data.

3.2 Causality and Statistical Dependence

Many researchers have associated the problem of causality with statistical dependence [Sup70], [Sky80] and [SGS90]. In this regard an important question is: How do we discover causal relations among a set of variables? We now consider in greater detail a relation (or a principle) which associates causality and statistical dependence. This is done by considering the possible causal directed acyclic graphs that could arise between a set of three variables.

To clarify the situation we list below an example taken directly from Verma *et al.*, [VP91]. Given three variables X, Y and Z, the set of causal dependency models that could exist between these variables are shown below[5].

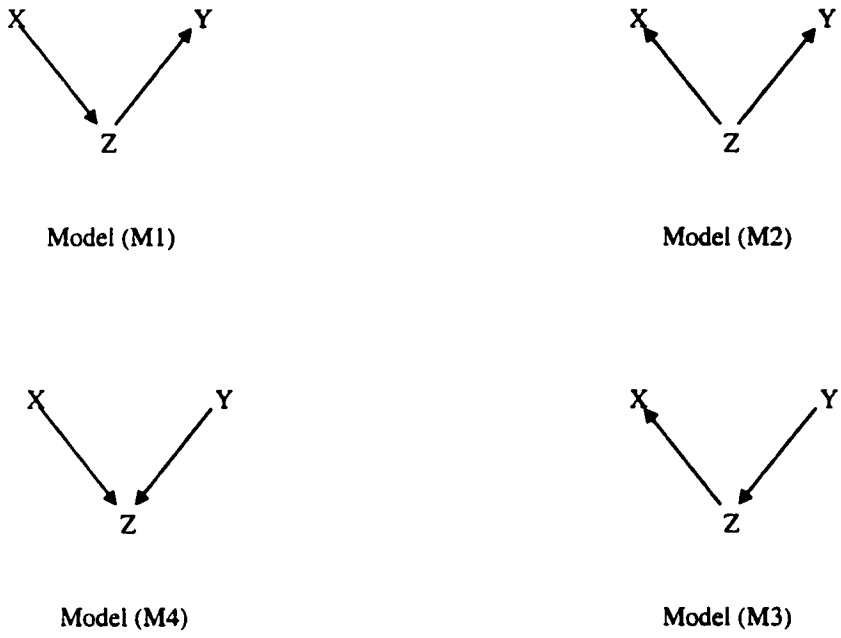


Figure 3.1: The causal models that can be obtained with three variables X, Y and Z.

Consider the dependency model (M1). The set of parameters portrayed by this model are: P(X), P(Z|X) and P(Y|Z). The set of parameters required for model (M2) are P(X|Z), P(Y|Z) and P(Z). For model (M3), the following probabilities required for its structure are: P(Y), P(Z|Y) and P(X|Z). It is easy to see that models (M1) and (M2) are equivalent because P(X) P(Z|X) = P(XZ) = P(Z) P(X|Z). Furthermore, model (M3) is also equivalent to these models since its parameters P(Y), P(Z|Y) and P(X|Z) can be obtained from those of models (M1) and (M2) [VP91]. However model (M4) is completely different from the previous models. Its parameters are P(X), P(Y) and P(Z|X,Y), and these parameters cannot be obtained from the sets of parameters of models (M1 – M3). The DAGs representing models (M1- M3) are indistinguishable in the sense that they carry the

[5] Note that Models M1 and M3 are the same except for the naming of the variables.

same set of independence assertions; I(X, Z, Y) (X and Y are conditionally independent given Z), while the DAG representing model (M4) is distinguishable from the previous DAGs because it represents the independence I(X, ∅, Y) which implies that X and Y are marginally independent which is a condition not represented in either of the former DAGs. In [RP87], the principle of model (M4) is used to orient the skeleton of the polytree.

3.3 Problem Statement

If we consider the polytree construction algorithm as given by [RP87] we observe that the order of traversing the tree in order to orient it is unspecified. The implementation strategy of determining the order of dependence tests is also left to the reader of that article. In this chapter we shall formally develop an algorithm which returns a polytree given the necessary inter-variable independence tests. The algorithm is similar, in principle, (and in philosophy) to the work of [RP87] except that we have formalized how the nodes have to be traversed and how the independence tests are to be invoked. This is the novel contribution of this work. Indeed, first of all, the algorithm determines the network structure or the tree using the MWST algorithm described earlier, and subsequently it utilizes the principle of model (M4) to orient the tree by beginning with the assumption that we have marginal independence between *at least* two parents of any node. Thus if, there is no independence of any two parents of a node the algorithm will fail stating that the underlying tree structure cannot be oriented to yield a polytree.

The problem of orienting the tree is done in two steps. The first step identifies all the independencies. Every two nodes X_i and X_j are independent if the following equality is satisfied:

$$P(X_i, X_j) = P(X_i) * P(X_j)$$

This equality is not always satisfied with sample data (in which case other independence tests will be used or a threshold value will be considered to determine the independence of two variables). But as this is still not of direct interest of this chapter we shall leave it for the present, and re-visit it in Chapter 4. Therefore, in this chapter, we assume that $P(X_i, X_j)$ will be exactly equal to $P(X_i) * P(X_j)$ if X_i and X_j are independent. We also

assume that we are provided with this information whenever it is requested. The second step is as follows: after inferring all the statistical independence between the pairs of variables, we use the principle of model (M4) for every triplet of variables X, Y and Z ordered as: X— Z — Y, and for such variables, we test for independence of X and Y. If X and Y are independent then X is a parent of Z and Y is a parent of Z. For any triplet X, Y and Z such that: X→ Z — Y, we test if X and Y are independent implying that Y is parent of Z otherwise Y is a child of Z. Utilizing all this information we shall show that the polytree can be efficiently computed if the underlying tree structure is systematically traversed.

3.4 A Depth First Search Algorithm for Building Polytrees

Our algorithm for inducing the polytree is an application of the depth-first search algorithm to causal components of the undirected tree.

Let T = (V, E) be a connected, undirected tree where V is the set of vertices, and E the set of edges. A vertex Z is said to be an *articulation point* vertices X and Y if we have independency between X and Y.

If we start from node X, all the causal[6] paths reached from X constitute what has been called a Causal Basin with connected components. As defined by Pearl [Pea88], a Causal Basin starts with a multi-parent cluster (a child node and all of its direct parents) and continues in the direction of causal flow to include all of the child's descendants and all of the direct parents of those descendants. An example of this is given in the following figure. Observe that the figure includes the polytree with all its causal basins.

[6] The term "causal" is used out of respect to the other researchers who have worked in this area. The question of whether one variable "causes" the second is more a philosophic question. We would like to avoid discussion on this issue in this thesis.

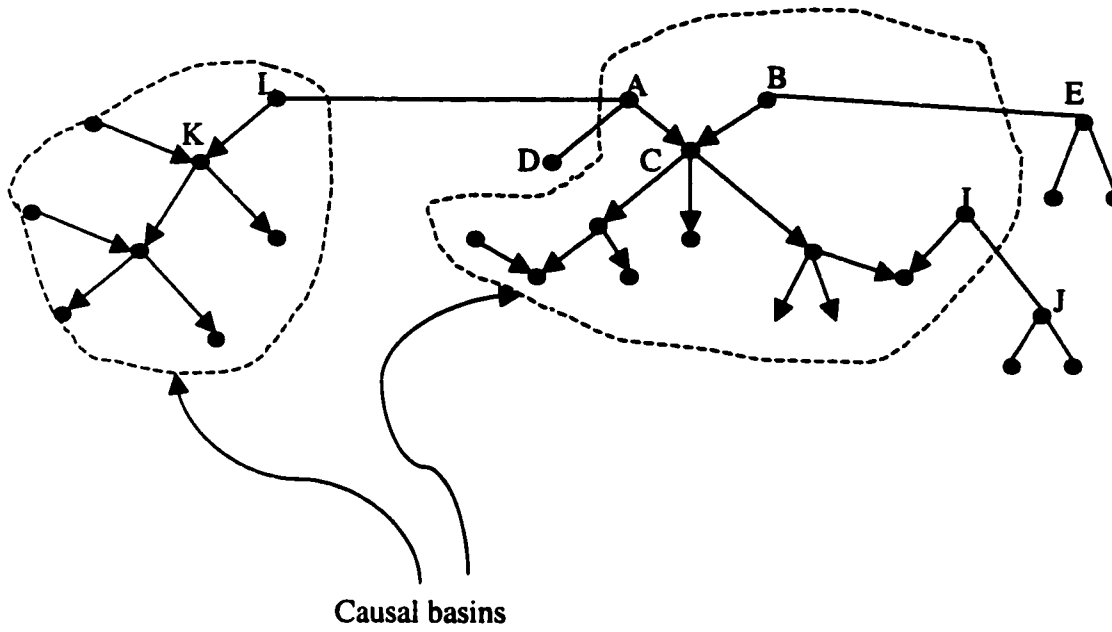


Figure 3.2: A Causal Basin as defined by Pearl [Pea88].

3.4.1 Problems with Pearl's Algorithm

Although the above definition is consistent, these are some unanswered questions which arise from the work of Rebane and Pearl [RP87]. In fact, although they specify a formal algorithm to compute the causal basins, they leave the following questions unanswered:

1. The question of what is meant by the outermost layer is not clear since it "depends on the tree" and its representation.
2. The question of how the traversal is done is not completely defined.
3. The algorithm introduces ambiguity regarding the edges that are already traversed.
4. The notion of causal basins depends on the starting point.

The last of these issues can be seen from the following figure in which the Chow tree of this polytree is taken from [Pea88].

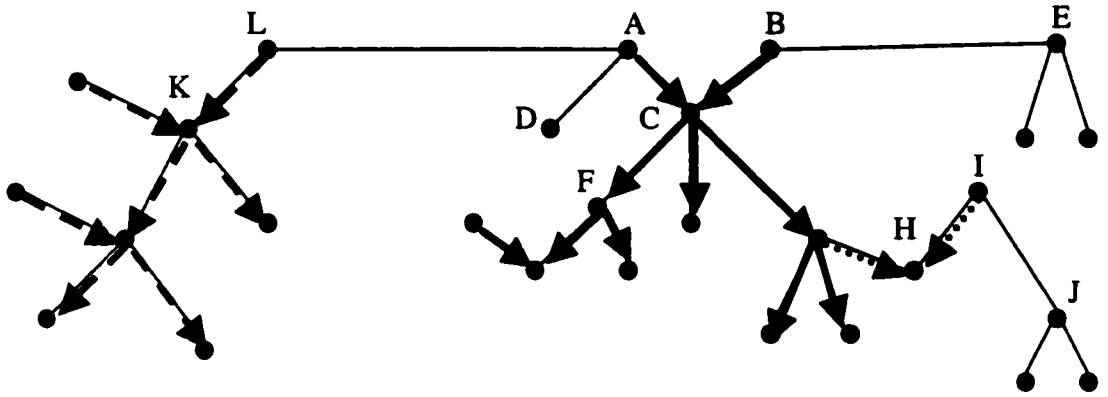


Figure 3.3: Three Causal Basins starting respectively at nodes H, K and C.

Observe that the Chow tree of Figure 3.2 and Figure 3.3 is the same. In Figure 3.2, the first articulation point (starting point) is node C and the second articulation point is node K. Having this order for the choice of the starting points we detect two causal basins as given in Figure 3.2. In Figure 3.3, we use node H as the first starting point, node C as the second and node K as the third to be able to complete the same orientation of the Chow tree as in Figure 3.2 using these causal basins instead of two. From this it is easy to see that the starting point determines the individual causal basins.

3.4.1 Motivation for a DFS Strategy

Consider the process of visiting the vertices of an undirected tree in the following manner. We select and "visit" a starting vertex Z which is one of the articulation points in T and in particular, an articulation point between two nodes X and Y. First of all we orient the edges (X, Z) and (Y, Z) as pointing to node Z following the orienting principle since we have independence between them. Then we select any edge (Z, W) incident upon Z. We check for independence between nodes X and W to determine the orientation of edge (Z, W). We observe two possible scenarios:

1. If there is no independence between X and W then edge (Z, W) is pointing to node W. We then visit node W and begin to search for a new edge starting at vertex W. After

completing the search through all causal paths beginning at W, the search returns either to Z, the vertex from which W was first reached, to search through all the adjacency list of Z, or to another non-visited articulation point.

2. If there is independence between X and W the edge (Z, W) is pointing to node Z. the search returns either to Z, the vertex from which W was first reached, to search through all the adjacency list of Z, or to another non-visited articulation point.

The process of selecting unexplored edges incident upon Z is continued until the list of these edges is exhausted. This method is summarized in the algorithm Polytree-Depth-First-Search.

The input to the algorithm is mainly the set of nodes, and for every node X_i we specify its "adjacency" list which consists of a list of nodes X_j such that arc X_i —X_j exists in the tree structure of the underlying tree. Also provided are the independence tests between nodes whenever required. The algorithm is formally given below.

Algorithm Polytree-Depth-First-Search

Input: A tree T=(V, E).
Independence test available for every pair of nodes when it is required.
For every node, a list of all its direct neighbors specified as a connected List specified as ConList. We assume that the test for a node being an articulation point is a straightforward operation. It essentially involves testing the condition of independence between its two neighbors.
Also, a node W is in the causal basin of X if there is a path from X to W.

Output: A directed polytree if the orientation exists. It returns "No" if any orientation is not possible.

Method

```
Begin
    For (all X in V) Do
        Visited [X] = false  /* Visited is an array holding the nodes */
    EndFor
        For (all X in V) Do
            /* Always starts with an articulation point */
            If ( !Visited[X] and X is an articulation point) Then
                Call Processing (X)
            EndIf
        EndFor
End Algorithm Polytree-Depth-First-Search


Procedure Processing (X)
Begin
    /* node X is not a leaf */
    If ((Visited[X] = false ) and (ConList(X)) > 1)) Then
        /* Orient the adjacent edges */
        Call IndepOrient(X)
        Visited[X] = true
        /*  traverse the adjacency list of X */
        For (all W in the ConList of X and W is in the causal basin of X) Do
            /* Processing is recursive because of Depth-First Search */
            Call Processing(W)
        EndFor
    EndIf
End Algorithm Processing
```

```
Procedure IndepOrient (X)
Begin
        For (every distinct N1 and N2 in ConList(X)) Do
                If Indep(N1, N2) = True Then
                        print arcs from (N1 to X) and (N2 to X)
                EndIf
        EndFor
        For (every distinct N1 and N2 in ConList(X)) Do
                If (arc from N1 to X exists and edge from N2 to X is unoriented) Then
                        print arc from X to N2
                Else If (arc from N2 to X exists and edge from N1 to X is unoriented)
Then
                        print arc from X to N1
                EndIf
        EndFor
End Algorithm IndepOrient
```

3.5 Convergence and Complexity of the Algorithm

The formal proof that the above algorithm terminates correctly follows the arguments of the Depth-First Search (DFS) traversal of a graph. But before we do it we shall give an example to make the traversal and the use of the independence tests clear.

Example 3.1:

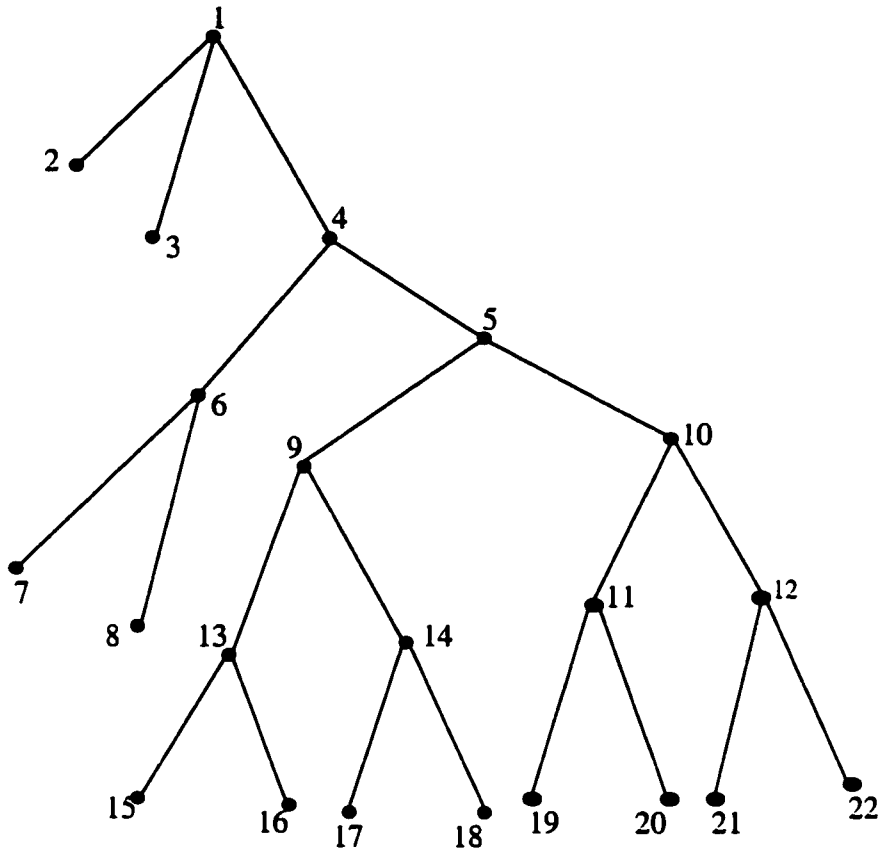


Figure 3.4: A Tree Example

In this example, we assume the independence information given in Table 3.1. This information may either be given *a priori* or obtained "on request". Also, for the sake of simplicity, we assume that the tree is formed of one causal basin.

I(2,3) = T	I(7,8) = F	I(17,9) = F	I(10,22) = F
I(2,4) = F	I(4, 9) = F	I(17,18) = F	I(10, 21) = F
I(3,4) = F	I(4,10) = F	I(12,5) = F	I(21, 22) = F
I(5,1) = F	I(9,10) = F	I(11,5) = F	
I(1,6) = F	I(9,16) = F	I(11, 12) = F	
I(5,6) = F	I(15,9) = F	I(10, 19) = F	
I(4,8) = F	I(15,16) = F	I(10, 20) = F	
I(4,7) = F	I(9,18) = F	I(19, 20) = F	

Table 3.1: Independence information about the nodes of the tree above.

As stated earlier, the starting point is always an articulation point. In Example 3.1 we assume that the DFS algorithm takes node 1 as the starting point to orient the tree since we have independence between nodes 2 and 3.

Step1: I(2, 3) = T. Therefore the edges are directed as:

Step2: The next independence tests between neighbors of 1 are:

I(3, 4) = F and I(2, 4) = F.

Since edge (2, 1) is already directed as pointing to 1, edge (1, 4) is directed in the causal basin of the edge (2, 1) as:

After directing all the edges starting at node 1, we mark it as a visited node and we progress in the DFS manner to the nodes belonging to its adjacency list.

Step 3: Both nodes 2 and 3 are marked as visited because they are leaves. We therefore move to node 4 and start the same process of testing independence between its neighbors as we did for node 1. We check for independence between every pair of its neighbors, which are:

I(6, 5), I(1, 5) and I(1, 6).

These tests fail and we know already that edge (1, 4) is pointing to node 4. Thus edges (4, 5) and (4, 6) are directed in the same causal flow as with edge (1, 4). The direction of the edges for the nodes processed till now is:

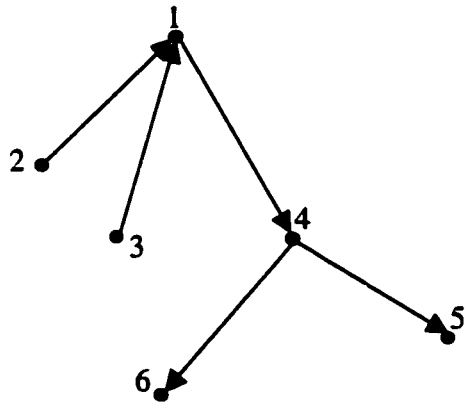


Figure 3.5: Portion of polytree oriented at the end of step 3.

Step4 - End: The same process will continue for all the nodes of the tree and all the edges will be directed in the same flow as they are in the same causal basin. The result of orienting the whole tree is given in Figure 3.6 below.

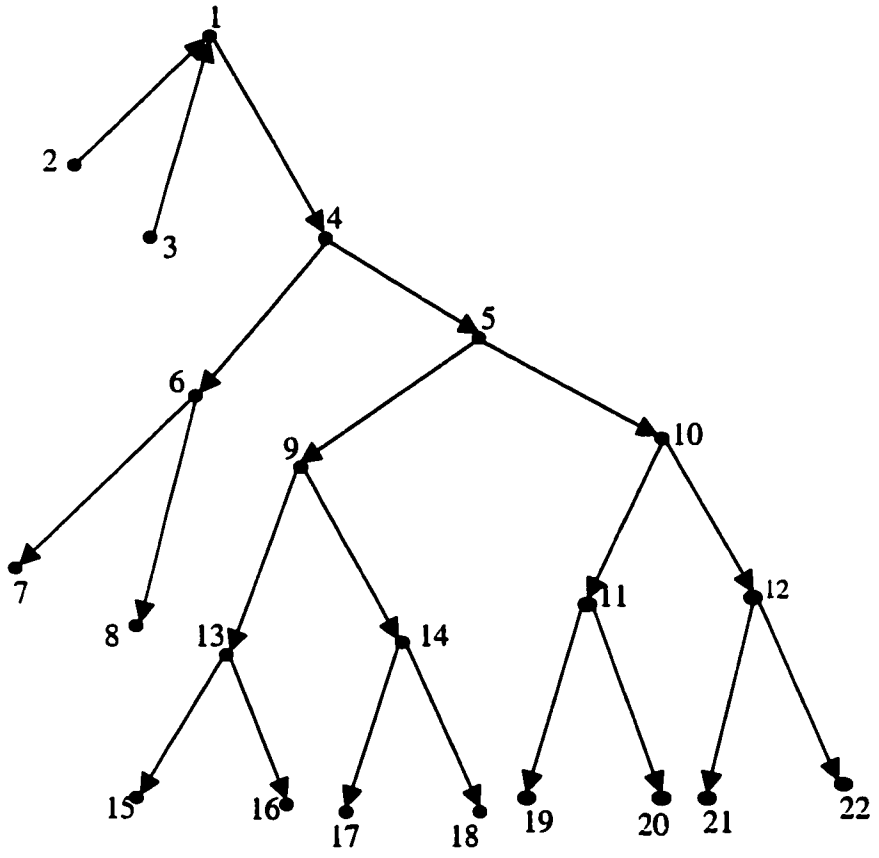


Figure 3.6: The Polytree obtained from tree in Figure 3.4 and the independence tests of Table 3.1. □

Theorem 3.1:

The algorithm Polytree-Depth-First-Search correctly orients the edges of the polytree corresponding to the skeleton tree structure and the underlying independence relationships.

Proof:

The formal proof of the correctness of the algorithm is now done by induction on the number of the nodes. We shall prove the following statements:

(i) Every node of the tree is visited, and,

(ii) Every edge in the tree is directed by the DFS as imposed by the ordering given by the independence tests.

Without loss of generality, we assume that we are processing one causal basin from the starting node. Thus if a new node is not visited it will be a starting point for another causal basin.

The first statement is trivial as a direct result of the DFS traversal to the tree. The more crucial argument involves the *orientation* of its edges.

Basis Step :

We use an articulation point as the starting point for the algorithm since it is clear that the algorithm will not lead to any orientation without such a starting point. The basis step involves the first articulation point X. Let the variables $\{X_i\}$ denote the neighbors of X and in particular, let X_n and X_m be the neighbors of X who satisfy $I(X_n, X_m) = T$ for indexes n and m because X is an articulation point. Thus edges (X_n, X) and (X, X_m) are oriented as pointing to node X since X_n and X_m are independent. We then proceed to check for independence between all pairs of the neighbors of X to orient the corresponding edges. If a test $I(X, X_j) = T$ for some j, it implies that the edge (X, X_j) points to node X; Otherwise it points to node X_j. This achieves the orientation of all the neighbors of X.

Inductive Hypothesis :

Suppose that at a certain stage of the algorithm we are visiting node Y, where node Y can be any node of the tree. Suppose that Y_c is one of the children of Y and Y_p is one of the parents of Y where the edges (Y, Y_c) and (Y, Y_p) are unoriented. When visiting node Y, we check for all the independence tests between the neighbors of Y which include Y_c and Y_p. We know that edge (Y_p, Y) is already oriented as pointing to Y since we have earlier visited Y_p. Two scenarios can happen:

(i) The first scenario is that the independence test $I(Y_p, Y_c)$ is false. In such a case we orient edge (Y, Y_c) as pointing to Y_c.

(ii) The second scenario is when the independence test $I(Y_p, Y_c)$ is true. In this case both edges (Y_p, Y) and (Y, Y_c) point to node Y.

By the same reasoning it is clear that we can keep orienting all the neighbors of Y, and using the DFS procedure we traverse the whole tree and orient all the edges. The result is thus true for all edges in one causal basin.

The result is also true if we have more than one causal basin because we would invoke the same DFS strategy from a new starting point in the next causal basin. Hence the theorem. □

The above algorithm uses a DFS strategy. It is easy to devise an analogous algorithm, which uses a Breadth-First search strategy, or for that matter, any systematic search scheme.

3.5.1 Complexity of the Algorithm

We shall now analyze the complexity of the above algorithm. This is done by considering the number of independence tests that need to be performed to be able to orient the tree. To do this we consider a tree T with n nodes and depth h where every node has (except the leaves) exactly k connected neighbors. Recall that in a tree structure every node has at most one parent and can have many children. In order to find the total number of independence tests we proceed by induction on the depth h of the tree. To clarify the computation we consider a specific example, which is a rather straightforward case of the corollary of Theorem 3.2 proved presently.

Example 3.3:

Consider the case of Figure 3.6 in which the number of nodes is 22. The root (node 1) has 3 children, and all the other nodes have only 2 children. Note that this exactly fits our model since every internal node has 3 dependent nodes – its parents and its two children, and thus, as mentioned in this case, $N = 22$ and $k = 3$.

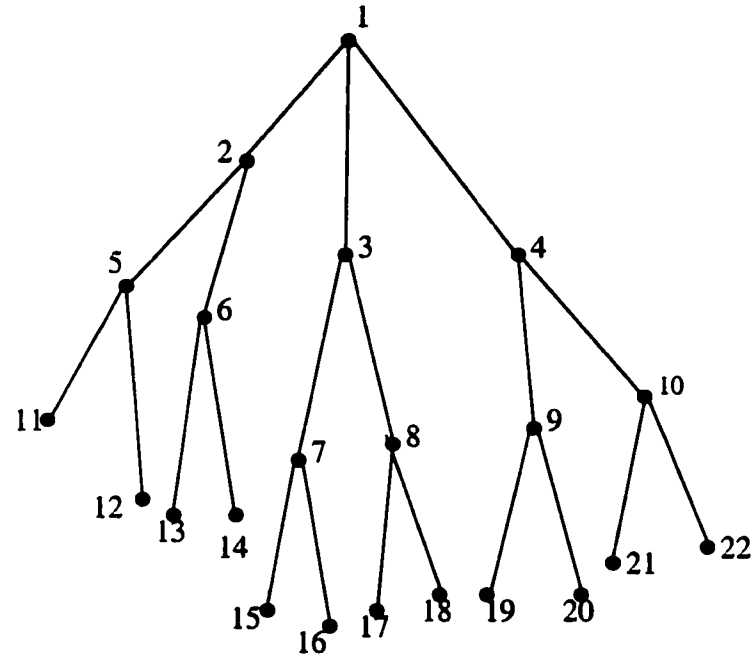


Figure 3.7: A tree example

We list below the number of independence tests, which need to be done for this tree in order to orient it.

Depth = 0: At this level three tests which must be done, namely, I(2,3), I(2,4), I(3,4)

which is $\binom{3}{2} = 3$ tests.

Depth = 1: At this depth the result of following tests which must be returned:

I(5, 6), I(5, 1), I(6, 1), I(7, 8), I(7, 1), I(8, 1), I(9,10), I(9,10), I(9,1)

equal to 9, which can be seen to be: $3 * \binom{3}{2} = 9$ tests.

Depth = 2: The tests in this case are:

I(11, 12), I(11, 2), I(12, 2), I(13, 14), I(13, 2), I(14, 2), I(15, 16), I(15, 3), I(16, 3), I(17, 18), I(17, 3), I(18, 3), I(19, 20), I(19, 4), I(20, 4), I(21, 22), I(21, 4), I(22, 4)

Which amount to 18 tests. It can be seen again that this is: $3*(3\text{-}1)^1 * \binom{3}{2} = 18$ tests.

Thus the total number of tests to be done is 30. The above results are formalized below.

Remark:

From a straightforward examination it is clear that the overall burden of the computation can be obtained by observing that for each node we have to do pairwise independence tests between its neighbors. Thus, in a straightforward manner the cost is $n * \binom{k}{2}$.

However, to understand what happens at every level of the tree, a more detailed study is needed. This is done in Theorem 3.2.

Theorem 3.2:

For a tree in which every node has up to k adjacent nodes, at depth **d** in the tree, the total number of independence tests, which have to be done, is:

$$k*(k-1)^{d-1}*\binom{k}{2}.$$

Proof:

The proof is done by induction.

Basis Step :

The basis step considers all nodes of depths 0, 1 and 2.

For **Depth 0**, the root has k dependents, and so we have to test every pair of nodes for independence. This results in $\binom{k}{2}$ tests $= \frac{1}{2}*k*(k-1)$ tests.

For **Depth 1**, every node in the k nodes has (k-1) dependents not counting the parent. The number of independence tests is $k*\binom{k-1}{2}$ tests for children and $k*(k-1)$ tests involving the nodes with the grandparent. This is equal to **Test(Depth 1)**, where

$$\textbf{Test(Depth 1)} = k*\binom{k-1}{2} + k*(k-1)$$

$$= \left[\frac{1}{2} * k * (k-1) * (k-2) + k * (k-1)\right] = k * \binom{k}{2}.$$

For **Depth 2**, every node in the $k*(k-1)$ nodes has k-1 dependents. The number of independence tests is **Test(Depth 2)** where

$$\textbf{Test(Depth 2)} = \left[k * (k-1) * \binom{k-1}{2} + k * (k-1) * (k-1)\right]$$

$$= k * (k-1) * \left[\frac{1}{2} * (k-1) * (k-2) + (k-1)\right]$$

$$= k * (k-1) * (k-1)\left[\frac{1}{2} * (k-2) + 1\right] = k^2 * (k-1)^2 = k * (k-1) * \binom{k}{2}.$$

As mentioned before, the expressions for d = 0, d =1 and d = 2 collectively constitute the basis case.

Inductive Hypothesis :

We now assume that the property is true until level d-1, i.e., the number of independence tests at depth d-1 is equal to **Test(Depth d-1)** where,

$$\textbf{Test(Depth d-1)} = k * (k-1)^{d-2} * \binom{k}{2}.$$

At depth d or layer d in the tree we have $k*(k-1)^{d-1}$ nodes each having (k-1) dependents not counting the parent. Once we are at depth d, we have to perform $\binom{k-1}{2}$ tests for every node at this depth, which amounts, for $k*(k-1)^{d-1} * \binom{k-1}{2}$ tests. Subsequently, we have to check every child of the $k*(k-1)^{d-1}$ nodes with its grand parent, which amounts for $k*(k-1)^{d-1}*(k-1)$ tests. Then the total number of tests is **Test(Depth d)** where,

$$\textbf{Test(Depth d)} = \left[k * (k-1)^{d-1} * \binom{k-1}{2} + k * (k-1)^{d-1} * (k-1)\right]$$

$$= k*(k-1)^{d-1}*\left[\binom{k-1}{2}+(k-1)\right] = k*(k-1)^{d-1}*\left[\frac{1}{2}+k*(k-1)\right]$$

$$= k*(k-1)^{d-1}*\binom{k}{2} = (k-1)*k*(k-1)^{d-2}*\binom{k}{2}$$

$$= k*(k-1)^{d-1}*\binom{k}{2}$$

The theorem follows. □

Corollary to Theorem 3.2:

As mentioned earlier, the total number of independence tests which needs to be done for a tree of depth d and branching factor k is $n*\binom{k}{2}$. This follows from the above theorem since:

$$\textbf{Total Tests} = \sum_{h=1}^{d} k(k-1)^{h-1}\binom{k}{2} = \binom{k}{2}*k*\sum_{h=1}^{d}(k-1)^{h-1}$$

But since we know that $\left[k*\sum_{h=1}^{d}(k-1)^{h-1}\right] = n$, we obtain,

$$\textbf{Total Tests} = \frac{1}{2}*k*(k-1)*n$$ □

3.6 Limitations

The limitations of the algorithm are primarily due to the lack of information about the independence between the variables. In order to orient the edges we used the independence tests between every two neighbors. This information may not be provided by the user and therefore it prevents the algorithm from orienting the tree. The example below illustrates the case. In this example, let us suppose that it is not possible to orient edges (F, G) and (E, G) because the dependence information about nodes E and F is not available. In a later section we shall present a strategy for providing a temporal ordering

of the variables and thus allowing an orientation following this ordering. Thus, if a variable A precedes a variable B in the "time ordering" we shall later only allow orientation of A towards B.

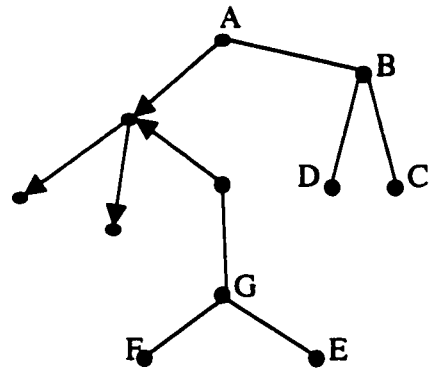


Figure 3.8: An example of an incomplete oriented polytree

3.7 The Polytree Algorithm

3.7.1 Discrete Case

Now that we have described the algorithm for traversing the tree and orienting the branches we shall proceed to the general overall algorithm. First we shall consider the case of discrete random variables where we assume that the probabilities are given in order to compute the MWST and we are not estimating them from samples. We also assume that the independence tests for the variables are available to the user (or system investigating the polytree structure). As we shall see in the experimental results presented in Section 3.8, when all the independence information is given, the algorithm completely orients the polytree following the Depth-First Search procedure mentioned above.

Algorithm Discrete-Polytree-Depth-First-Search

Input: The probabilities $P(X_i, X_j)$ for all pairs of random variables X_i, X_j.

Output: The dependence polytree, which utilizes only these second, order statistics and satisfy the I_f or the I_χ measure.

Method

Begin

For (i =1 to N and j < i) **Do**

Compute $I_f(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) \log \frac{P(X_i, X_j)}{P(X_i)P_t(X_j)}$ **OR**

Compute $I_\chi(X_i, X_j) = \sum_{X_i, X_j} \frac{(P(X_i, X_j) - P(X_i)P(X_j))^2}{P(X_i)P(X_j)}$

EndFor

Repeat Until A Spanning Tree T is completed

Include the branch with largest weight unless a cycle is obtained.

If cycle is obtained **Then**

Discard edge.

EndIf

EndRepeat

Call Polytree-Depth-First-Search (T) to obtain the polytree

End Algorithm Discrete-Polytree-Depth-First-Search

3.7.2 Continuous Case

Since we have fully described the discrete case, it is appropriate in the interest of completeness to consider case of the continuous variables where the vector $\underline{X} = [X_1, X_2, ..., X_N]^T$ is jointly normal with mean vector $\underline{\mu} = [\mu_1, \mu_2, ..., \mu_N]^T$ and the covariance matrix $\Sigma = [\sigma_{ij}]$. The algorithm of traversing the tree is the same as in the discrete case. The algorithm is formally given below without further explanation.

Algorithm Normal-Polytree-Depth-First-Search

Input: The parameters $\underline{\mu}$ and Σ for the normal distribution obeyed by $\underline{X}$:

Output: The best dependence polytree obtained using the I_f or I_χ measure.

Method

Begin

Compute R:= $[\rho_{ij}]$, the matrix of correlation coefficients from Σ.

For (i=1 to N and j<i) **Do**

Compute W_{ij} = $I_f(X_i, X_j) = -\frac{1}{2}\log(1 - \rho_{ij})$ **OR**

$$I_\chi(X_i, X_j) = \frac{\rho_{ij}^2}{1 - \rho_{ij}^2}.$$

EndFor

Obtain the maximum weight spanning tree τ^* from the vector of nodes and { W_{ij} }

Call Polytree-Depth-First-Search (τ^*) to obtain the polytree

End Algorithm Normal- Polytree-Depth-First-Search

3.8 Experimental Results

In order to demonstrate the power of the algorithm developed in this chapter we have done numerous experiments. We assumed that we were to learn an underlying polytree, which is unknown to the algorithm. One such polytree is given in Figure 3.9. Also, we assumed that independence tests, which are consistent with the polytree orientation, were available whenever they were needed by the algorithm. These independence tests for the specific polytree in Figure 3.9 are given in Table 3.2. The polytree learning algorithm *Discrete-Polytree-Depth-First-Search* was invoked using the skeleton and the sequence of independence tests. The results of running the polytree algorithm demonstrate its power.

Since we already know the distribution of the data, we first run the Chow algorithm using the given joint distributions. We then obtain the skeletal tree of the polytree, the tree given in Figure 3.10. We are thus required to orient this tree using the independence tests given in Table 3.2. In order to orient it the algorithm detected the following articulation points: nodes 2, 4, 9 and 11 as starting points for different causal

basins. The algorithm then attempted to orient all the separate causal basins starting at the respective articulation points, which represent nodes with multiple parents as specified by the independence tests. After running the algorithm, the orientation of every edge of the "true" underlying polytree was compared with the orientation of the edge of the resultant "inferred" polytree. In every case the results are exactly the same. These experiments were conducted for various kinds of polytrees, namely polytrees with one or multiple causal basins. In every case, the polytree was exactly inferred.

A second example is given is given in Figures 3.11 and 3.12. Its independence information is given in Table 3.3.

Our experimental results consistently demonstrate that the algorithm successfully orients the polytree. Again we observe that if the independence information for any pair of nodes is not provided to the algorithm, it will fail to orient the entire tree.

I(0,1) = T	I(3,4) = F	I(10,13) = F
I(2,9) = T	I(5,6) = F	I(10,12) = F
I(7,8) = T	I(5,9) = F	
I(12,13) = T	I(9,6) = F	
I(1,3) = F	I(7,11) = F	
I(0,4) = F	I(4,8) = F	
I(2,6) = F	I(4,7) = F	
I(2,5) = F	I(9,10) = F	

Table 3.2: Independence information for the polytree given in Figure 3.9.

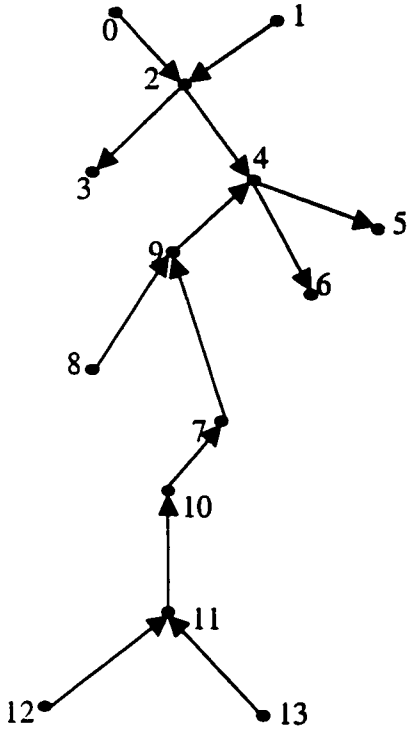


Figure 3.9: Underlying dependence polytree

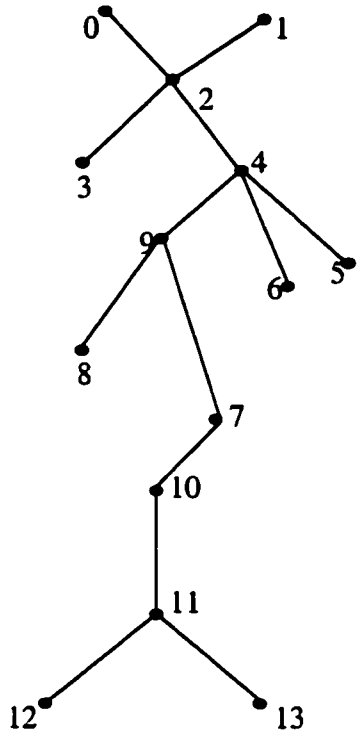


Figure 3.10: Tree structure T1

I(11,10) = T	I(3,1) = F
I(10,8) = F	I(4,1) = F
I(11,8) = F	I(2,5) = F
I(9,5) = F	
I(1,8) = T	
I(8,7) = F	
I(8,6) = F	
I(3,4) = T	

Table 3.3: Independence information for the polytree given in Figure 3.11.

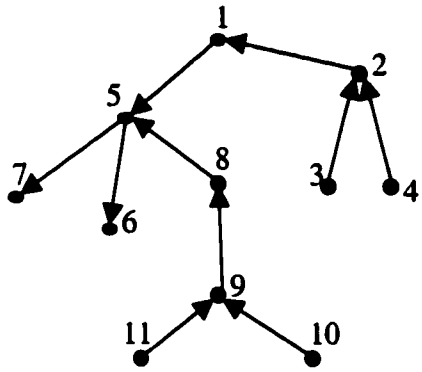


Figure 3.11: Underlying dependence polytree

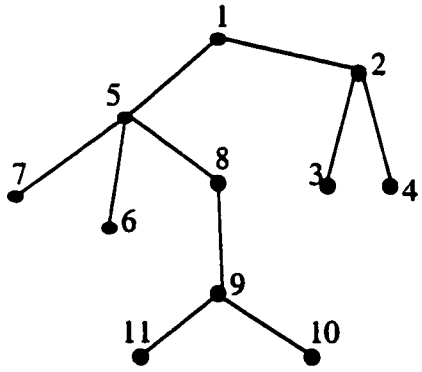


Figure 3.12: The skeletal tree of the underlying polytree given in Figure 3.11.

3.9 Conclusion

In this chapter we have considered the problem of determining an approximate polytree-structured distribution for an underlying distribution. The skeletal form of the polytree is known to be the MWST of a complete graph, with $I_f(X_i, X_j)$, the information theoretic metric, as the edge weight between the pair of nodes X_i and X_j. Once the tree is derived, Rebane and Pearl, in [RP87] used an independence test to determine if a variable has multiple parents. In fact, they dictated that every two node neighbors X and Y of a node Z

are considered and tested to see if they are marginally independent, and hence to decide if node Z has parents X and Y.

This chapter proposes a formal and systematic algorithm, which traverses the tree obtained by the Chow method, and using the independence tests it successfully orients the polytree. Our algorithm uses an application of the DFS strategy to multiple causal basins. It first efficiently detects the articulation points as the starting points for any causal basin. It then traverses every causal basin and computes the orientation of its edges. Apart from describing the algorithm and proving its correctness, we have also derived the closed form expression for its complexity. Indeed we have given an expression for the maximum number of independence tests required to complete the orientation of the polytree. We have also implemented the algorithm and tested it on numerous polytrees.

Experimental results demonstrate that when the required independence tests are available to the algorithm, the orientation of the polytree is completed. Furthermore, in all the tests performed, the underlying orientation was always correctly detected. Also, the algorithm requires at least one starting point (an articulation point) which, in terms of independence, implies that at least two nodes are needed to completely orient the polytree.

The advantages of our algorithm are numerous; first of all it is computationally efficient since it uses a DFS scheme. It also extends itself to both discrete and continuous variables, and gives a very efficient way of traversing the tree. Finally we should mention that the scheme also handles multi-feature variables.

A summary of the results of this chapter is currently being compiled as a publication [OOMA99].

Chapter 4

Generating Sample Data from a Directed Acyclic Graph

4.1 Introduction

In learning probabilistic models, learning algorithms can be applied to determine the structure and the probability parameters describing the model, if the training samples are given. In several learning applications it is often necessary to approximate probability distributions/densities. In estimating this probability distribution, it is assumed that no information about the probability is available. One such estimator is the polytree method given in [RP88] and described in the previous chapter, and it is clear that we need to test and compare estimators before we use them in real-life.

In this chapter we shall consider the problem of generating sample data from a given structure. The reasons for studying this are the following. First of all, the problem is interesting in its own right. Secondly, in order to test the validity of the various methods developed for the construction of a polytree from data, we are faced with the fundamental problem of randomly generating data obeying an underlying polytree structure. Thus, we need random numbers and vectors following tightly constrained relations obtained from dependence structures.

In this chapter, we describe a method for generating random samples from a joint probability distribution. The method is general, in the sense that it can be applied to any Directed Acyclic Graph (DAG). In this section we shall study how can we obtain a *distribution* based on which we can generate random data that satisfies a given polytree structure. In other words we intend to obtain a valid *distribution* that the polytree structure will permit. To the best of our knowledge the question of generating such a distribution has not been reported in the literature and this is the primary contribution of this chapter.

4.2 Related Work

4.2.1 Henrion's Work

A related work is given by Henrion in [Hen88] in which the author introduces the concept of probabilistic logic sampling. The probabilistic logic sampling algorithm assumes a Bayesian network with prior distributions specified for all the source variables and conditional distributions for the other nodes. It initially uses a random number generator to produce a sample truth value for the source variables, and a sample *implication* rule for each parameter of each conditional distribution using the corresponding probabilities. It subsequently repeats the same process through the network following the arrows (paths) from the source nodes using simple logical operations to obtain the truth values of each of the variable from its parents and the implication rules [Hen88]. The difference between Henrion's [Hen88] work and ours is that we do not invoke logical operations, but rather get rigid probability constraints to yield the possible random vectors that can be generated from the polytree structure. The problem is more complicated than the one addressed in [Hen88] because the only information available to us is the given structure, and no other information about the distribution itself is available. For the case of known probabilities we can have different scenarios.

4.2.2 The Case of Known Conditionals

If the first order probabilities for sources, and the conditionals[7] for the children are given, it is easy to see that the random values of the samples can be generated in a very straightforward manner. An example will clarify this.

[7] The program K2 [CH92] assumes that there is a distribution on the conditionals as an added constraint.

Example 4.1:

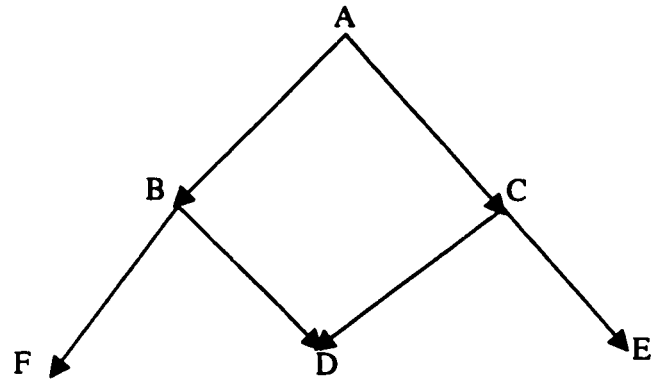


Figure 4.1: An Example of a Bayesian belief network

Given are:

P(A=0) =	0.7
P(B=0|A=0) =	0.45
P(B=0|A=1) =	0.3
P(C=0|A=0) =	0.9
P(C=0|A=1) =	0.75
P(E=0|C=0) =	0.4
P(E=0|C=1) =	0.1
P(D=0|B=0, C=0)=	0.23
P(D=0|B=1, C=0) =	0.6
P(D=0|B=1, C=1)=	0.9
P(D=0|B=0, C=1)=	0.5
P(F=0|B=0) =	0.53
P(F=0|B=1) =	0.68

What is derived are thus:

P(A=1) =	0.3
P(B=1|A=0) =	0.55
P(B=1|A=1) =	0.7
P(C=1|A=0)=	0.1
P(C=1|A=1) =	0.25
P(E=1|C=0) =	0.6
P(E=1|C=1) =	0.9
P(D=1|B=0, C=0) =	0.77
P(D=1|B=1, C=0) =	0.4
P(D=1|B=1, C=1) =	0.1

P(D=1|B=0, C=1) = 0.5
P(F=1|B=0) = 0.47
P(F=1|B=1) = 0.32

In Figure 4.1, the influence of node A on B and C is quantified by the conditional probability distributions, P(B|A) and P(C|A). The influence of variables B and C on D is expressed by the conditional distribution, P(D|B,C). The influence of variables C on E is expressed by the conditional distribution, P(E|C) and the influence of variable B on F is given by P(F|B). The present method assumes the knowledge of P(A), P(B|A), P(C|A), P(D|B, C) , P(E|C) and P(F|B). Let us suppose that we know P(A=0). Thus P(A=1) is deduced. By invoking one random number call, we can generate randomly the values of A. If now we know that A=0, we utilize P(B=0|A=0), (also indirectly P(B=1|A=0)) and invoke a random number call again to get the value of B. The same process is invoked for the other conditionals. Note that the number of parameter variables for the nodes is high, since we need parameters for the sources and conditionals for all the nodes with children. For binary variables, if the number of parents of a node X_i is k, it requires a table of size 2^k to express all the conditional probabilities for node X_i.

This result may be generalized using the theorem of Pearl [Pea88] (see Theorem 9) which was discussed in the context of building Bayesian networks. He described the task of building BBNs as the following:

- Structuring the network
- Quantifying the links.

The task of structuring the network for a given probabilistic model and a node ordering of the variables was solved in [PV87] and in [VP88]. This was discussed in detail as a related work in Chapter 2, Section 2.2 and given as Verma/Pearl algorithm for building a BBN. To specify the strengths of the influences that the parents have on the children in a DAG, Pearl assessed the conditional probabilities $P(X_i|Parents(X_i))$ by some functions $F_i(X_i, Parents(X_i))$ such that:

$$\sum_{X_i} F_i(X_i, Parents(X_i)) = 1 \text{ where } 0 \leq F_i(X_i, Parents(X_i)) \leq 1.$$

Thus, if the prior probabilities for the sources are known, and the conditional distributions $P(X_i|Parents(X_i))$ on the links of the DAG are given, the above formula can be used to generate random occurrences of the vector in a straightforward manner.

4.2.3 The Case for Known First Order Probabilities

In the case studied in the previous subsection, it was shown that we need to specify 2^k parameters for every node having k children. However, if the first order probabilities are known (which is a more realistic case), we shall show that we need to specify only N parameters for the same number of nodes (N is the number of nodes in the BBN), and we can now show that all possible conditional distributions are not permitted. For example if $P(X_0= 0) = 0.396$ and $P(X_1= 0) = 0.840$, the conditional $P(X_1= 0|X_0= 0)$ cannot be assigned arbitrary value. It can be seen that the latter probability follows some constraints, as it is a joint probability of two events. Indeed, as we shall see, generating random vectors in this scenario is a much harder problem than the case studied in 4.2.2, and to the best of our knowledge the problem has not been addressed before. We believe that this case is a more realistic scenario since these first order probabilities can be reliably estimated from samples in a real-life situation.

Our contribution is the generation algorithm, which we intend to develop. First it yields the joint probabilities associated with the various components of the vector $\underline{X} = [X_1, X_2,..., X_N]^T$ where X_1, X_2, ... and X_N represent the nodes of a Directed Acyclic Graph (DAG) using which the specific values of $\{X_i\}$ can be generated. The question we therefore study is that of obtaining samples of a vector $\underline{X}$ whose marginal distributions are specified, and whose dependence relationships are described by a DAG. We shall consider in greater detail the discrete case. Our aim is to devise an algorithm that allows us to generate a set of samples or a set of instances for the vector $\underline{X} = [X_1, X_2,..., X_N]^T$ where X_1, X_2,..., and X_N are Boolean random variables. This process will create input data that we can use to test the algorithms already developed in earlier chapters. Also these samples are generated following a Directed Acyclic Graph (DAG) structure constraint. This sample generation algorithm is achieved following these three steps:

(i) The first step of the algorithm is to generate randomly a distribution for N random variables.

(ii) The second step uses the network *structure* as a constraint to produce the set of prior probabilities to all source variables, and conditional probabilities to the rest of the variables.

(iii) The last step determines the sample vector, which is a vector of values of the N variables from the conditional probabilities. This procedure consists of generating the truth-value for every variable using its probability.

Figure 4.2 is an example of a DAG representing a probabilistic causal network for diagnosing patients with lung diseases. This example is taken from [BKRK97]. The variables can take on one of the values *True* or *False*.

Given the network of Figure 4.2, which is represented as a DAG, we will describe how we can derive the joint probabilities for the nodes of this network. This is done in Section 4.1. Following then the structure of the network, we derive conditional probabilities for the nodes of the network using the structure constraint and finally derive the samples for the vector:

$\underline{X}$ = [Smoker, CoalMiner, LungDisease, PositiveXray, Dyspnea, Emphysema]^T.

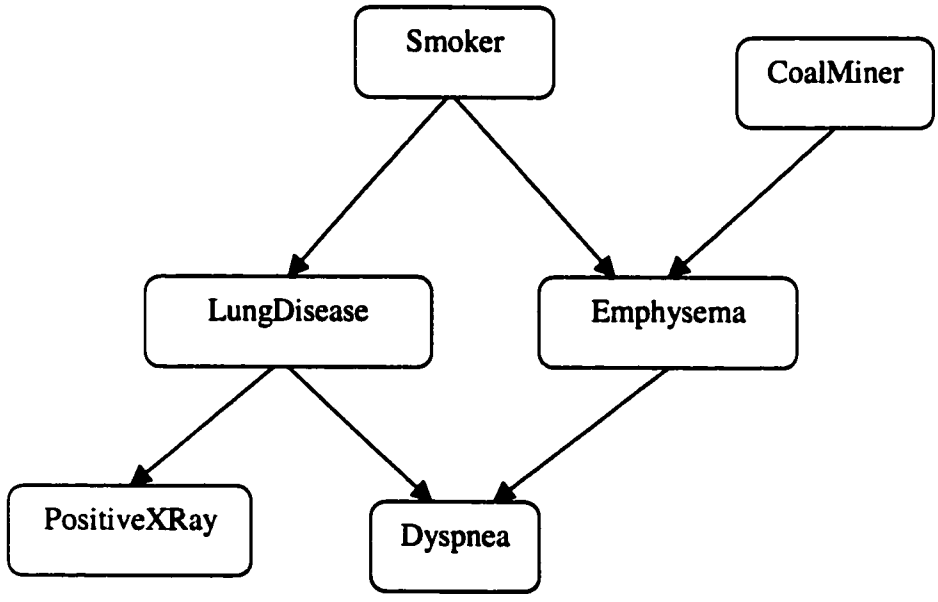


Figure 4.2: Probabilistic Network Describing Patients ([BKRK97])

4.2.4 Covariance and Mean Known for Normal Distributions

If the mean μ and the covariance matrix Σ of the underlying normal distribution are given, the question of generating random vectors with the distribution $N(\underline{\mu}, \Sigma)$ where the variables dependence is a polytree based dependence is described in Section 4.6.

4.3 Generate Probabilities

The first task of our algorithm is to generate the joint probabilities. For the first order probabilities we generate[8] random variables distributed uniformly between constants 0 and 1. We assign uniformly values to $P(X_i = 0)$ between 0 and 1, and thus implicitly assign values to $P(X_i = 1)$ for all the variables. We also generate randomly probabilities of higher order uniformly between two bounds namely *low* and *high*. These bounds have to be determined from a set of constraints described in the next sections. In [VO92], the authors generated sample data following a tree structure. They considered the computation of first and second order marginals since, in a tree every node has at most one parent. They generated second order probabilities using constraints or inequalities given in (4.1) below.

For any two Boolean random variables X and Y, P(X,Y) should be chosen randomly following:

$$\text{Max}\,(0, P(X) + P(Y)-1) \leq P(X,Y)) \leq \text{Min}\,(P(X), P(Y)) \qquad (4.1)$$

In this thesis, we are faced with the crucial problem of studying how to compute the probabilities of order higher than two. To derive this we present the following algorithms and theorems.

[8] In a real life scenario these first order probabilities will be typically provided.

4.3.1 Generating all Valid Sequences[9]

Before computing the probabilities, we first generate the sequences of all possible values, which the variables can take. Indeed, we construct a k^{th} sequence in terms of the $(k-1)^{st}$ sequences preceding it. We recursively compute the sequences with the algorithm Compute-Sequences given below written in a recursive procedural-like manner.

Algorithm Compute-Sequences
Input : Sequences for order k =1 which are 0 and 1.
Output : Sequences for order k.
Method
Begin
Compute-Sequences(1) ← {0, 1}.
Compute-Sequences(k, NewSet) ←
Compute-Sequences(k-1, Set),
Reverse(Set,TempSet),
Prefix-0-Sequences(Set),
Prefix-1-Sequences(TempSet),
Concatenate(Set,TempSet, NewSet).
End Algorithm Compute-Sequences

The procedure Prefix-0-Sequences adds the prefix '0' to all sequences in Set, the procedure Reverse yields the list Set in the reverse order and store these sequences in TempSet. The procedure Prefix-1-Sequences prefixes by the digit '1' every sequence in the variable TempSet. The procedure Concatenate merge both lists Set and TempSet to obtain the list of sequences for order k called NewSet. An example to explain this process is given below.

[9] It will be apparent to the reader that the sequence generated falls within the family of Grey Code sequences, typically used in coding theory. What we present below is the actual pseudo-code to generate one specific Grey Code which can be used in a formal algorithm to generate random variables values and to test their consistency.

Example 4.2 : We shall show how the set is generated for k = 3.

For order k = 1, sequences(1) = {0, 1}.

For order k = 2, we take sequences of order 1 given in Set = {0, 1}, TempSet = reverse(Set) = {1, 0}.

We prefix every sequence in Set with the digit '0' and rewrite Set as = {00, 01}.

We prefix every sequence in TempSet by the digit '1'and rewrite TempSet as = {11, 10}.

Thus NewSet = {00, 01, 11, 10}

For order k = 3, since sequences(2) = {00, 01,11,10}, we take the reverse of Set and store it in TempSet as = {10, 11, 01, 00}, we prefix by '0' every sequence in Set and rewrite Set as = {000, 001, 011, 010}. We now prefix by digit '1' every sequence of TempSet and rewrite TempSet as = {110, 111, 101, 100}.

Finally NewSet = Set + TempSet = {000, 001, 011, 010, 110, 111, 101, 100}. □

The equivalent code[10] in a Prolog-like language for algorithm Compute-Sequences is given below. It is included because it is easier to implement and is more space efficient.

```
seq(1, [0,1] :-!
seq(N, L) :-                    N1 is N-1, seq(N1, L1),
                                reverse(L1, LRev),
                                pad(0,L1,L2),
                                pad(1,LRev, LRev1),
                                append(L2, LRev1, L).

reverse(L, LRev) :-             reverse(L, LRev, []).
reverse([ ], L , L).
reverse([H|T], L, L1) :-        reverse(T, L, [H|L1],

string_append(X, Y, XY) :-      name(X, NX),
                                name(Y, NY),
                                append(NX, NY, NXY),
                                name(XY, NXY).

pad(Ch, [ ], [ ]).
pad(Ch, [H|T], [H1|T1]) :-      string_append(Ch, H, H1), pad(Ch, T, T1).
```

[10] Please see the footnote at the sub-section heading 4.3.1.

We shall now prove the properties of the algorithm Compute-Sequences. The proof is quite straightforward, but included in the interest of completeness.

Theorem 4.1: The algorithm computes all the 2^k sequences of order k with the additional property that every sequence and its neighbors differ only in one bit.

Proof :

The proof of the theorem is completed in two parts and both are done by induction on k.

Proof (a) : All the 2^k sequences are generated.

Basis Step :

For k =1, two sequences are generated which are 0 and 1. Clearly all the 2^1 are now generated.

Inductive Hypothesis :

We assume that for order k-1, all 2^{k-1} sequences are generated and stored in the variable Set. The set of sequences we generate for order k will be NewSet. This set is obtained by :

(i) first including the sequences for order k-1, prefixed by a '0'.

(ii) taking all the sequences in Set in reverse order and prefixing every sequence by the digit '1' to obtain the sequences stored in TempSet.

The cardinality of Set is 2^{k-1} by the induction hypothesis. The number of sequences formed in NewSet is exactly the number of sequences formed in Set added to the number of sequences generated in TempSet. We can thus write the following equality :

cardinality(NewSet) = cardinality(Set) + cardinality(TempSet)

$= 2^{k-1} + 2^{k-1} = 2^k$.

Hence the result. □

Proof (b) : Every two consecutive sequences in the generated set differ only in one bit.

Basis Step :

For k =1, the two sequences which are generated are 0 and 1, which clearly differ in only one bit.

Inductive Hypothesis :

We assume that for order k-1, all 2^{k-1} sequences differ only in one bit.

In order to prove that every two consecutive sequences differ only in one bit for order k, we consider two consecutive sequences S_1 and S_2 and we analyze the following cases.

Case 1: S_1 and $S_2 \in$ Set.

S_1 and S_2 differ only in one bit because they represent two sequences of order k-1 which already differ only in one bit by the inductive hypothesis and which we prefixed by the digit '0'.

Case 2: S_1 and $S_2 \in$ TempSet.

S_1 and S_2 differ only in one bit since all sequences in TempSet are exactly the sequences of Set taken in reverse order where we have padded the digit '1' to the (k-1) order sequences. Again the inductive hypothesis is sufficient to show that S_1 and S_2 differ only in one bit.

Case 3: $S_1 \in$ Set and $S_2 \in$ TempSet.

This case can only be possible if S_1 and S_2 are in the boundaries, i.e., S_1 is the last sequence in Set and S_2 is the first sequence in TempSet. When we reverse the list Set the last element of Set will be the first element of TempSet. Furthermore this last element of Set will be prefixed by '0'. We therefore conclude that since S_2 is prefixed with '1' and S_1 is prefixed by '0' the two differ in only this bit, since the rest of the sequences are identical. Thus the theorem is proved. □

The reader should observe that:

- The sequences fall within the family of Grey Codes. The difference is that we have given a specific representation and a generation strategy.
- The distance between the subsequent sequences is unity. This implies that the set of sequences minimizes the Hamming distance consecutively. We therefore refer to this particular sequence generated by algorithm Compute-Sequences (a member of the Grey Code family) as *a Distance Diminishing Sequence (DDS).*

Once the sequences are generated we compute the probabilities associated with these sequences. These probabilities are computed in the same order as the sequences are obtained from the algorithm Compute-Sequences. To compute the joint probabilities of order k, we first generate sequences of length k and use the probabilities already computed of order k-1. The following theorem shows how this process is completed.

Theorem 4.2: Given the joint distribution of all (k-1) random variables, and the sequences of random variables of order k as a Distance Diminishing Sequence (DDS), the computation of the distribution for k random variables requires only one random number call. Furthermore, in this case the k^{th} order probability can be computed by visiting each k^{th} order sequence exactly once.

Proof :

The proof is completed by induction on the index of the sequence visited in the DDS. We assume that we invoke a random number generator to get the probability $P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=0)$, which is the probability associated with the sequence $\underline{a}$ = <0 0 0....0>. This probability will be chosen randomly between certains bounds that will be discussed in the next section.

Basis Step :

We can get the probability $P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=1)$ which is the probability associated to the sequence <0 0 0....1>. We know that sequence <0 0 0 ..1> is the next sequence in the DDS after <0 0 0...0>. Therefore these two sequences differ only in one bit, namely the last bit. Consequently,

$$P(X_0=0, X_1=0, X_2=0,\ldots..., X_{k-1}=0) = P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=0) + P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=1)$$

Since the left hand side is known from the inductive hypothesis and the first term in the right hand side is known from the random number call, we get

$$P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=1) = P(X_0=0, X_1=0, X_2=0,\ldots..., X_{k-1}=0) - P(X_0=0, X_1=0, X_2=0,\ldots..., X_k=0)$$

Inductive Hypothesis :

Assume that the probabilities associated to the first j sequences in the DDS have been computed.

We can compute the probability for the $(j+1)^{th}$ sequence which differs from the j^{th} sequence at bit X_t as follows :

Let the j^{th} sequence be :

$$\underline{a} = (X_1 = b_1, X_2 = b_2, \ldots, X_t = b_t, \ldots, X_k = b_k), \text{ and the } j+1^{st} \text{ sequence be :}$$

$$\underline{b} = (X_1 = b_1, X_2 = b_2, \ldots, X_t = \bar{b}_t, \ldots, X_k = b_k).$$

a and b differ only in the t^{th} bit.

$$P(X_1 = b_1, X_2 = b_2, \ldots, X_{t-1} = b_{t-1}, X_{t+1} = b_{t+1} \ldots, X_k = b_k) =$$

$$P(X_1 = b_1, X_2 = b_2, \ldots, X_t = b_t, \ldots, X_k = b_k)$$

$$+ \quad P(X_1 = b_1, X_2 = b_2, \ldots, X_t = \bar{b}_t, \ldots, X_k = b_k)$$

Since the left hand side is known from the inductive hypothesis, and the first term in the right hand side is known from the random number call, we get

$$P(X_1 = b_1, X_2 = b_2, \ldots, X_t = \bar{b}_t, \ldots, X_k = b_k) =$$

$$P(X_1 = b_1, X_2 = b_2, \ldots, X_{t-1} = b_{t-1}, X_{t+1} = b_{t+1} \ldots, X_k = b_k)$$

$$- \quad P(X_1 = b_1, X_2 = b_2, \ldots, X_t = b_t, \ldots, X_k = b_k).$$

Thus the theorem is proved. □

4.3.2 Bounding the probabilities when using the random calls

In the previous section we showed how to compute the k^{th} order probabilities for all the instances of the variables of the vector $\underline{X}$. We assumed that we invoked a random number generator to get the probability $P(X_0=0, X_1=0, X_2=0, \ldots, X_k=0)$, which is the probability associated with the sequence <0 0 0....0>, and once we have this probability, all the others are derived using Theorem 4.2. We studied the problem of computing these probabilities using the DDS. In order to compute $P(X_0=0, X_1=0, X_2=0, \ldots, X_k=0)$ for all values of k such that $k \geq 2$, we need to find the constraints or the bounds because these probabilities cannot be arbitrarily chosen, and this should be done for different values of

k. Indeed, we need the intervals from which we are allowed to choose these random numbers calls. The following theorems show how these bounds are determined.

Theorem 4.3:

The generation process for obtaining the probabilities for n variables invokes $2^n - 1$ distinct equations, which, in turn, yield 2^n-1 distinct inequalities, which bound the corresponding probabilities.

Proof:

The proof is done by an enumeration strategy based on the number of variables n.

Let us denote by $P_{ijk....}^{ABC...}$ the joint probability P(A = i, B = j, C = k, ...) where A, B and C are variables and i, j and k are values of these random variables respectively. In our case where A, B and C are binary variables i, j and k take values 0 or 1. We also denote by P_{ij}^{AB} the joint probability of two variables A and B taking respectively values A = i and B = j, and also P_i^A represents the marginal probability of variable A taking value A = i.

To initiate arguments we consider the elementary cases when n =1, 2 and 3. Based on these, the arguments for n> 3 will be better appreciated.

For **n = 1**, we have one variable A and we know that $P_0^A + P_1^A = 1$ (4.2)

P_0^A can be chosen between 0 and 1, and so $P_1^A = 1 - P_0^A$.

For **n =2**, we have two variables A and B. We observe that we have these equations:

$$P_{00}^{AB} + P_{01}^{AB} + P_{10}^{AB} + P_{11}^{AB} = 1 \quad (4.3)$$

Also $$P_{00}^{AB} + P_{01}^{AB} = P_0^A \quad (4.4)$$

$$P_{00}^{AB} + P_{10}^{AB} = P_0^B . \quad (4.5)$$

Observe that from these three equations we have to compute four probabilities. Observe too that if we generate randomly P_{00}^{AB} between certain bounds, we can subsequently compute the other three using the axioms of probability. From these equations we deduce:

$P_{01}^{AB} = P_0^A - P_{00}^{AB}$ and $P_{10}^{AB} = P_0^B - P_{00}^{AB}$.

We know that $P_{01}^{AB} \geq 0$ and $P_{10}^{AB} \geq 0$, and so,

$P_{00}^{AB} \leq P_0^A$ and $P_{00}^{AB} \leq P_0^B$ which means

$P_{00}^{AB} \leq \min(P_0^A, P_0^B)$.

From the equation (4.3) we see that:

$P_{11}^{AB} = 1 - P_{00}^{AB} - P_{01}^{AB} - P_{10}^{AB}$, and $P_{11}^{AB} \geq 0$.

Thus, $1 - P_{00}^{AB} - P_{01}^{AB} - P_{10}^{AB} \geq 0$.

If we replace

P_{01}^{AB} by $P_{01}^{AB} = P_0^A - P_{00}^{AB}$, and

P_{10}^{AB} by $P_{10}^{AB} = P_0^B - P_{00}^{AB}$,

we get the following:

$$1 - P_0^A + P_{00}^{AB} - P_0^B \geq 0,$$

which means $P_{00}^{AB} \geq P_0^A + P_0^B - 1$ and $P_{00}^{AB} \geq 0$ therefore:

$$P_{00}^{AB} \geq \max(0, (P_0^A + P_0^B - 1)).$$

To compute P_{00}^{AB} we thus need to invoke a number generator between the following bounds $P_{00}^{AB} \geq \max(0, (P_0^A + P_0^B - 1))$ and $P_{00}^{AB} \leq \min(P_0^A, P_0^B)$, and so we deduce the other probabilities using the previous theorems.

Consider the case of three variables (n = 3). We know from the axioms of probability theory that the following equations are true:

$$P_{000}^{ABC} + P_{001}^{ABC} + P_{010}^{ABC} + P_{011}^{ABC} + P_{100}^{ABC} + P_{101}^{ABC} + P_{110}^{ABC} + P_{111}^{ABC} = 1 \quad (4.6)$$

$$P_{000}^{ABC} + P_{001}^{ABC} + P_{010}^{ABC} + P_{011}^{ABC} = P_0^A \quad (4.7)$$

$$P_{000}^{ABC} + P_{001}^{ABC} + P_{100}^{ABC} + P_{101}^{ABC} = P_0^B \quad (4.8)$$

$$P_{000}^{ABC} + P_{010}^{ABC} + P_{100}^{ABC} + P_{110}^{ABC} = P_0^C \quad (4.9)$$

$$P_{000}^{ABC} + P_{001}^{ABC} = P_{00}^{AB} \quad (4.10)$$

$$P_{000}^{ABC} + P_{100}^{ABC} = P_{00}^{BC} \quad (4.11)$$

$$P_{000}^{ABC} + P_{010}^{ABC} = P_{00}^{AC} \quad (4.12)$$

The problem now is to find a bound or range in which we can randomly assign a value to P_{000}^{ABC}. The rest of the other probabilities are deduced as follows which is exactly the enumeration of the theorem on how to compute the probabilities using the sequences.

$$P_{001}^{ABC} = P_{00}^{AB} - P_{000}^{ABC}$$

$$P_{011}^{ABC} = P_{01}^{AC} - P_{001}^{ABC}$$

$$P_{010}^{ABC} = P_{01}^{AB} - P_{011}^{ABC}$$

$$P_{110}^{ABC} = P_{10}^{BC} - P_{010}^{ABC}$$

$$P_{100}^{ABC} = P_{10}^{AC} - P_{110}^{ABC}$$

$$P_{101}^{ABC} = P_{10}^{AB} - P_{100}^{ABC}$$

$$P_{111}^{ABC} = P_{11}^{AC} - P_{101}^{ABC}$$

To define P_{000}^{ABC}, the bounds are computed from the previous seven equations and are as follows:

$$P_{000}^{ABC} \leq P_{00}^{AB}$$

$$P_{000}^{ABC} \leq P_{00}^{BC}$$

$P_{000}^{ABC} \leq P_{00}^{AC}$. This means that:

$$P_{000}^{ABC} \leq \min (P_{00}^{AB}, P_{00}^{BC}, P_{00}^{AC}). \qquad (4.13)$$

Similarly, $P_{000}^{ABC} \geq P_{00}^{AB} + P_{00}^{BC} - P_{0}^{B}$

$$P_{000}^{ABC} \geq P_{00}^{AC} + P_{00}^{BC} - P_{0}^{C}$$

$P_{000}^{ABC} \geq P_{00}^{AB} + P_{00}^{AC} - P_{0}^{A}$, and thus,

$$P_{000}^{ABC} \geq \max (P_{00}^{AB} + P_{00}^{BC} - P_{0}^{B}, P_{00}^{AC} + P_{00}^{BC} - P_{0}^{C}, P_{00}^{AB} + P_{00}^{AC} - P_{0}^{A}). \qquad (4.14)$$

Also, from the general equation summing all the third order probabilities to unity we derive the following inequality for P_{000}^{ABC}:

$$P_{000}^{ABC} \leq 1 - P_{0}^{A} - P_{0}^{B} - P_{0}^{C} + P_{00}^{AB} + P_{00}^{AC} + P_{00}^{BC}. \qquad (4.15)$$

Thus, P_{000}^{ABC} should be randomly chosen between the bounds defined by the inequalities (4.13), (4.14) and (4.15).

Arguing as in the above, it is easy to see that, to generate the probabilities for j variables we can derive $1 + \sum_{k=1}^{j-1} \binom{j}{k} = 2^j - 1$ equations. These equations are obtained as:

1 equation summing all the probabilities to unity.

$\binom{j}{1}$ equations giving the equalities for probabilities of order 1, and in general,

$\binom{j}{i}$ equations giving the equalities for probabilities of order i.

Thus the total number of equalities is:

$$1+\binom{j}{1}+\binom{j}{2}+.....+\binom{j}{j-1} = 2^j - 1 \text{ equations,}$$

and the theorem is proved. □

Remark:

In practice it can be seen that the set of inequalities derived may be lead to "contradictory" conclusions. In other words, it is possible that there is no region in [0,1] which satisfies all of them. This will thus mean that a distribution cannot be obtained under these conditions. An example for this situation is given below.

Example 4.3:

Assume that we are dealing with a problem having three variables A, B and C. We first randomly assign values to P_0^A, P_0^B and P_0^C between 0 and 1 as:

$P_0^A = 0.396$, $P_0^B = 0.840$ and $P_0^C = 0.318$.

For second order probabilities we have to generate randomly values for P_{00}^{AB}, P_{00}^{BC} and P_{00}^{AC} between the bound defined above. To be more specific we shall clearly present the bounds in which these probabilities must lie. From previous equalities (4.3), (4.4) and (4.5), we have the following:

$P_{00}^{AB} \leq \min(P_0^A, P_0^B)$ and $P_{00}^{AB} \geq \max(0, P_0^A + P_0^B - 1)$ which yields:

$P_{00}^{AB} \leq \min(0.396, 0.840)$, (4.16)

$P_{00}^{BC} \leq \min(0.840, 0.318)$ and (4.17)

$P_{00}^{AC} \leq \min(0.396, 0.318)$. (4.18)

$P_{00}^{AB} \geq \max(0, 0.236)$, $P_{00}^{BC} \geq \max(0, 0.158)$ and $P_{00}^{AC} \geq \max(0, -0.286)$.

P_{00}^{AB} must be randomly obtained from (0.236, 0.396),

P_{00}^{BC} must be randomly obtained from (0.158, 0.318) and

P_{00}^{AC} must be randomly obtained from (0, 0.318)

We assume that the random number calls for these are as follows:

$P_{00}^{AB} = 0.239$, $P_{00}^{BC} = 0.185$ and

$P_{00}^{AC} = 0.122$.

For third order probabilities we will do the same process; We first define the bounds for P_{000}^{ABC} and subsequently generate the rest of the probabilities.

We have $P_{000}^{ABC} \leq \min(0.239, 0.185, 0.122)$

and $P_{000}^{ABC} \geq \max(-0.416, -0.011, -0.035)$.

Also P_{000}^{ABC} should satisfy the following:

$P_{000}^{ABC} \leq 1 - P_0^A - P_0^B - P_0^C + P_{00}^{AB} + P_{00}^{AC} + P_{00}^{BC}$ then $P_{000}^{ABC} \leq -0.008$.

This inequality is not possible and therefore P_{000}^{ABC} cannot be assigned a value, implying that one cannot derive a valid distribution with the above first order and second order probabilities assignments.

Theorem 4.4:

To obtain the values of all the 2^n probabilities, the generation process requires $2^n - 1$ random number calls.

Proof:

We show that in order to generate the 2^n probabilities for n variables we need 2^n-1 random number calls.

This proof shows that the 2^n probabilities for n variables can be uniquely assigned by exactly $= \sum_{i=1}^{n} \binom{n}{i}$ random number calls. The proof is done by an enumeration based on the number of variables n. For n variables we will have to compute the following:

The first order probabilities for all the variables, and these make n random calls.

The joint probabilities for all possible combinations of two variables which makes:

$\binom{n}{2}$ **random calls.**

The joint probabilities for all possible combinations of j variables will make $\binom{n}{j}$ **random calls.**

Also we have to make one random call to generate the joint probability of all the n variables implying **one more random call.**

We conclude that: for n variables the total of random calls we make is:

$$(n) + \binom{n}{2} + \binom{n}{3} + \ldots + \binom{n}{k} + 1 =$$

$$= 2^n - 1 \text{ random number calls.}$$

This proves the theorem. □

4.4 Generate Conditional Probabilities from the Network

In order to generate the conditional probability from a given network, we exploit the given *structure* of the network to derive the conditional probabilities on the nodes of the network. For the nodes which do not have any parents, nodes which are called "source" nodes or leaves we use the first order probabilities to generate the samples. We call a random number with $P_r(X_i = 1)$ to assign a truth value to X_i in the sample. For the other nodes, we generate the conditional probabilities on the nodes in such a way that the

parent is assigned a probability first before the child is assigned any probability. In fact, for every node X_i whose parents are given by $Par(X_i) = \{X_{i1}, X_{i2},..., X_{im}\}$, we compute

$$P_r(X_i | Par(X_i)) = P_r(X_i | (X_{i1}, X_{i2},..., X_{im})) = \frac{P_r(X_i, X_{i1}, X_{i2},..., X_{im})}{P_r(X_{i1}, X_{i2},..., X_{im})} \quad (4.19)$$

In the case of polytree structure which is a special case of a DAG where the parents of a node are marginally independent, we have the following equality:

$$P_r(X_{i1}, X_{i2},..., X_{im}) = \prod_{k=1}^{m} P_r(X_{ik})$$

which simplifies the formula above (4.19) and yields the following:

$$P_r(X_i | (X_{i1}, X_{i2},..., X_{im})) = \frac{P_r(X_i, X_{i1}, X_{i2},..., X_{im})}{\prod_{k=1}^{m} P_r(X_{ik})} \quad (4.20)$$

If we deal with DAGs then for every node we compute its conditional probability given by :

$$P_r(X_i | (X_{i1}, X_{i2},..., X_{im})) = \frac{P_r(X_i, X_{i1}, X_{i2},..., X_{im})}{P_r(X_{i1}, X_{i2},..., X_{im})}$$

We can say that this procedure for generating samples can be used to generate samples from an underlying polytree or an underlying Bayesian belief network, where the condition of independence of parents of a given node will be used in the case of polytrees.

4.5 Generate Samples

Once we have the conditional probabilities we call a random number generator with the probability on the node to produce the truth-value of the node in the sample vector. This leads us to the following algorithm to generate samples following the three steps we have already described.

Algorithm Generate-Samples

Input: A DAG structure
Output: A random distribution obeying the network structure and a set of samples or values for the vector $\underline{X} = [X_1, X_2, ..., X_N]^T$.

Method
Begin
Generate randomly first order probabilities $P(X_i)$ for all source variables between constants 0 and 1.
For (all the remaining nodes) **Do**
If ($Par(X_i) = \{X_{i1}, X_{i2}, ..., X_{im}\}$) **Then**
Compute bounds for $P(X_i=0\ X_{i1}=0\ X_{i2}=0..., X_{im}=0)$.
Assign the probability within the bounds and derive the distribution for node X_i.
Call a number random generator with the distribution of X_i to obtain its truth-value in the vector $\underline{X}$.
EndIf
EndFor
End Algorithm Generate-Samples

4.6 Generating Samples from Continuous Variables

In this final subsection we shall address the problem of generating samples from an underlying normal distribution whose parameters μ and Σ are known. Before we proceed to determine the samples for a given continuous distribution, we establish the notion of polytree dependence in continuous distribution. We assume that the random vector $\underline{X} = [X_1, X_2, ..., X_N]^T$ is jointly normal with mean $\underline{\mu} = [\mu_1, \mu_2, ..., \mu_N]^T$ and the covariance matrix $\Sigma = [\sigma_{ij}]$. The marginal density of the random variable X_i has the uni-variate normal density given by:

$$P(X_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\{-\frac{1}{2}\left(\frac{X_i - \mu_i}{\sigma_i}\right)^2\}$$

The distribution of $\underline{X}$ is denoted as $N(\underline{\mu}, \Sigma)$ and the density function $P(\underline{X})$ is given by:

$$P(\underline{X}) = (2\pi)^{-N/2} |\Sigma|^{-0.5} \exp\{-\frac{1}{2}(\underline{X} - \underline{\mu})^T \Sigma^{-1}(\underline{X} - \underline{\mu})\}.$$

As in the discrete case, this joint distribution is said to be of *polytree dependence* if $P(\underline{X})$ can be written in the form:

$$P(\underline{X}) = \prod_{i=1}^{N} P(X_i \mid X_{j1}(i), X_{j2}(i),..., X_{jm}(i)),$$

where $X_{j1}(i)$, $X_{j2}(i)$,....,$X_{jm}(i)$ are parents of X_i and are statistically independent. Consequently,

$$P(X_{j1}(i), X_{j2}(i),....,X_{jm}(i)) = \prod_{j=1}^{m} P(X_{ij}(i)) \text{ for all i.}$$

Therefore $P(\underline{X}) = \prod_{i=1}^{N} P(X_i \mid X_{j1}(i), X_{j2}(i),..., X_{jm}(i))$ because the right hand side of this equation represents the conditional density of the random variable X_i, conditioned on the set of its parents, then:

$$P(\underline{X}) = \prod_{i=1}^{N} \frac{P(X_i, X_{j1}(i), X_{j2}(i),..., X_{jm}(i))}{P(X_{j1}(i))P(X_{j2}(i)),..., P(X_{jm}(i))}$$

The question of generating random vectors with the distribution $N(\underline{\mu}, \Sigma)$ where the variables dependence is a tree based dependence is given in [VO92] and it is based on the following properties:

Property 1: let $\underline{X}$ be the vector $\underline{X} = [X_1, X_2,..., X_N]^T$ which has the distribution $N(\mu, \Sigma)$. Then the random vector $\underline{Y} = A\underline{X}$ has the distribution $N(A\mu, A\Sigma A^T)$. □

Property 2: Let μ, Σ be the mean vector and the covariance matrix of a normal distribution respectively. Let $\underline{Z}$ be a normal random vector which has the distribution N(0, I). Also, let $\Lambda = \text{diag}(\lambda_1, \lambda_2,..... \lambda_N)$ and Φ be the diagonal eigenvalue matrix, and the eigenvector matrix of Σ respectively. Then

$$\underline{X} = \mu + \Phi\Lambda^{1/2}\underline{Z}$$

has the normal distribution $N(\mu, \Sigma)$. □

The algorithm, which generates the samples from a multi-variate Normal distribution, is described below. It assumes that the function "GetUnivariateNormal" for generating normal variates with the distribution N(0, I) is available to the user. The procedure "GetEigenValues" and "GetEigenVectors" is also assumed to be available.

Algorithm Generate-Samples-Normal (μ, Σ, X)

Input: The parameters μ and Σ for the normal distribution of $\underline{X}$.
Output: Samples S drawn from the underlying normal distribution.
Method

Obtain Eigen Values and Vectors (Σ,Φ, Λ)
For (i=1 to N) **Do**
Z[I] = GetUnivariateNormal
EndFor
Compute $X = \mu+\Phi\Lambda^{1/2} Z$

End Algorithm Generate-Samples-Normal

The proof that the algorithm is correct is quite straightforward and is based on Property 1 and Property 2. The same algorithm will be used to generate samples where there is an underlying tree dependence for the variables. The question of imposing polytree dependence on these normal random variables seems to be unsolved.

4.7 Experimental Results

The following experimental results illustrate the use of the theorems presented in this chapter. For simplicity we assume that the variables are binary valued. We study an example of a DAG representing a probabilistic network for diagnosing patients who have diabetes as given in Figure 4.3. This example is taken from [Die97]. As the reader will observe, we have changed the type of variables to be binary to fit them in our scheme. There are six variables named:

(X_0) Age: True if the age of patient is higher than 65;

(X_1) Pregnancies: True if the patient had multiple pregnancies;

(X_2) Mass: True if mass is larger than 100kg;

(X_3) Insulin: True if blood-insulin level is high;

(X_4) Diabetes: True if patient has diabetes;

(X_5) Glucose: True if high level of glucose in the blood.

In the first step of the experiment we derive the joint probabilities for the nodes of the network shown in Figure 4.3. Using the structure constraint we compute the samples for the vector:

$\underline{X} = [Age, Pregnancies, Mass, Diabetes, Insulin, Glucose]^T$.

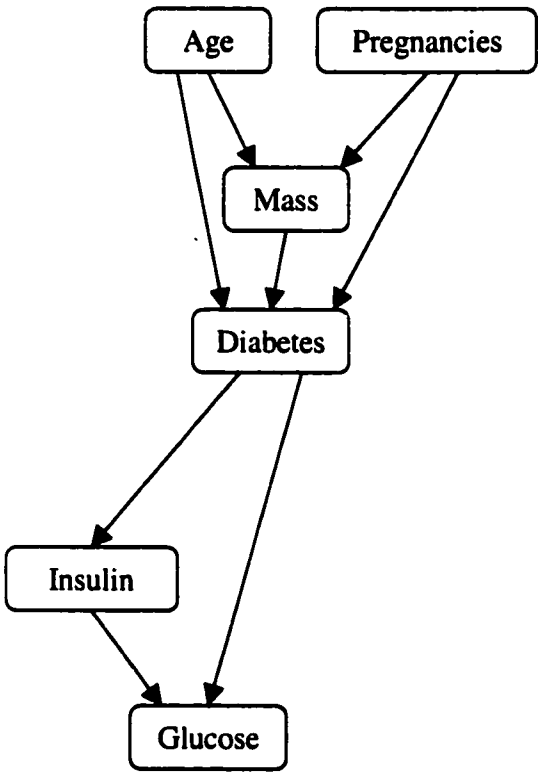


Figure 4.3: An Example of a Bayesian Belief Network (taken from [Die97])

The following are the joint probabilities for $\underline{X}$ = [Age, Pregnancies, Mass, Diabetes, Insulin, Glucose]^T = $[X_0, X_1, X_2, X_3, X_4, X_5]^T$

First order marginals

For first order probabilities, we uniformly assign values to $P(X_0= 0)$, $P(X_1= 0)$, $P(X_2= 0)$, $P(X_3= 0)$, $P(X_4= 0)$ and $P(X_5= 0)$ between 0 and 1. From these values we compute the probabilities for the other values of the variables and obtain the results given in Table 4.1:

$P(X_i=0)$	$P(X_i=1)$
$P(X_0=0) = 0.997012$	$P(X_0=1) = 0.002988$
$P(X_1=0) = 0.756727$	$P(X_1=1) = 0.243273$
$P(X_2=0) = 0.992061$	$P(X_2=1) = 0.007939$
$P(X_3=0) = 0.477978$	$P(X_3=1) = 0.522022$
$P(X_4=0) = 0.871383$	$P(X_4=1) = 0.128617$
$P(X_5=0) = 0.510040$	$P(X_5=1) = 0.489960$

Table 4.1: First Order Probabilities

Second order marginals

For second order probabilities we derived bounds specified by the previous theorems and using these bounds, we randomly assign a value to the joint probability $P(X_i=0, X_j=0)$ for all the variables. Once this choice has been made, the remaining second-order probabilities can be deduced using Theorem 4.2. The bounds *low* and *high*, which are computed for the example of Figure 4.3, are given in Table 4.2. To get the joint distribution for all the random variables, the same process is repeated for all pairwise combinations of variables. The following table is the result for the second order marginals.

(X_i,X_j)	*Low*	*High*	$P(X_i=0, X_j=0)$
X_0, X_1	0.754	0.757	0.755
X_0, X_2	0.989	0.992	0.989
X_0, X_3	0.475	0.478	0.478
X_0, X_4	0.868	0.871	0.869
X_0, X_5	0.507	0.510	0.507
X_1, X_2	0.749	0.757	0.756
X_1, X_3	0.235	0.478	0.381
X_1, X_4	0.628	0.757	0.736
X_1, X_5	0.267	0.510	0.456
X_2, X_3	0.470	0.478	0.474
X_2, X_4	0.863	0.871	0.866
X_2, X_5	0.502	0.510	0.503
X_3, X_4	0.349	0.478	0.427
X_3, X_5	0.000	0.478	0.229
X_4, X_5	0.381	0.510	0.500

Table 4.2: Second Order Marginals

In order to compute the third order probabilities, using the previous theorems, we computed the bounds from which we assign a probability to $P(X_i=0,X_j=0, X_k=0)$ for all values of i, j and k. From these probabilities we derive all the joint probabilities for all possible values of X_i , X_j and X_k. The results obtained for the upper and lower bounds are given below.

Third order marginals

(X_i, X_j, X_k)	*Low*	*High*	$P(X_i=0,X_j=0, X_k=0)$
X_0, X_1, X_2	0.754	0.754	0.754
X_0, X_1, X_3	0.381	0.381	0.381
X_0, X_1, X_4	0.734	0.735	0.734
X_0, X_1, X_5	0.454	0.454	0.454
X_0, X_2, X_3	0.474	0.474	0.474
X_0, X_2, X_4	0.864	0.864	0.864
X_0, X_2, X_5	0.500	0.500	0.500
X_0, X_3, X_4	0.426	0.427	0.426
X_0, X_3, X_5	0.229	0.229	0.229
X_0, X_4, X_5	0.498	0.498	0.498
X_1, X_2, X_3	0.380	0.381	0.381
X_1, X_2, X_4	0.735	0.736	0.736
X_1, X_2, X_5	0.455	0.456	0.456
X_1, X_3, X_4	0.360	0.381	0.366
X_1, X_3, X_5	0.176	0.229	0.209
X_1, X_4, X_5	0.447	0.456	0.455
X_2, X_3, X_4	0.423	0.426	0.424
X_2, X_3, X_5	0.226	0.226	0.226
X_2, X_4, X_5	0.495	0.495	0.495
X_3, X_4, X_5	0.219	0.229	0.228

Table 4.3: Third Order Probabilities

After computing the third order probabilities, we repeat the same process to compute the fourth order probabilities and the results are given in Table 4.4. These probabilities are necessary to generate the samples, which follow the structure given in Figure 4.3. In fact, we compute the conditionals from the joint probabilities of different order and therefore generate the samples. Notice that in the case of the third column of the diagram shown in

Figure 4.3, we are actually creating cases of patients fitting the structure of the diagram. The meaning of the variables used is:

$$\underline{X} = \text{[Age, Pregnancies, Mass, Diabetes, Insulin, Glucose]}^T$$

$$= [X_0, X_1, X_2, X_3, X_4, X_5]^T$$

Also the process of generating samples following the underlying structure can be used to synthetically obtain statistics for patients with the symptoms given in $\underline{X}$. These cases are quite hard to get in real-life.

Fourth order marginals

(X_i, X_j, X_k, X_l)	*Low*	*High*	$P(X_i=0, X_j=0, X_k=0, X_l=0)$
X_0, X_1, X_2, X_3	0.380	0.380	0.380
X_0, X_1, X_2, X_4	0.734	0.734	0.734
X_0, X_1, X_2, X_5	0.454	0.454	0.454
X_0, X_1, X_3, X_4	0.366	0.366	0.366
X_0, X_1, X_3, X_5	0.209	0.209	0.209
X_0, X_1, X_4, X_5	0.453	0.453	0.453
X_0, X_2, X_3, X_4	0.424	0.424	0.424
X_0, X_2, X_3, X_5	0.226	0.226	0.226
X_0, X_2, X_4, X_5	0.493	0.493	0.493
X_0, X_3, X_4, X_5	0.228	0.228	0.228
X_1, X_2, X_3, X_4	0.366	0.366	0.366
X_1, X_2, X_3, X_5	0.209	0.209	0.209
X_1, X_2, X_4, X_5	0.454	0.455	0.454
X_1, X_3, X_4, X_5	0.208	0.209	0.209
X_2, X_3, X_4, X_5	0.225	0.225	0.225

Table 4.4: Fourth Order Marginals

4.8 Summary

In this chapter, we studied the problem of generating sample data from an underlying structure. The problem is important because its solution will help the user to synthetically generate samples from an underlying polytree structure, which is particularly crucial, in scenarios whenever real-life data is hard to get. Thus, the user will be able to get input data for various polytree algorithms (including those introduced by us) with samples, which are randomly generated, from a true polytree, and to test the algorithms, which we have devised for both the discrete and continuous cases.

In this chapter we first showed how to generate a *distribution* for N Boolean variables and using this we showed how we could derive a method to build samples associated with the underlying structure. The only restriction made on this structure is that it is a DAG. This is the primary contribution of this chapter.

In order to compute the joint probabilities, $P(X_0=0, X_1=0, X_2=0,....., X_k=0)$ for all possible values of k and all values of X_k, we showed that we need to find the intervals or the bounds from which we choose the probability $P(X_0=0, X_1=0, X_2=0,....., X_k=0)$, because these probabilities cannot be arbitrarily chosen. We derived the constraints that allow us to determine the bounds. Once the probabilities $P(X_0=0, X_1=0, X_2=0,....., X_k=0)$ are generated randomly, between the bounds already determined for all values of k, we demonstrated how we would compute all the other relevant probabilities for all the values of X_k. This was shown in Theorem 4.2.

We also proposed a method to derive the probabilities and presented the resultant constraints from which these probabilities should be chosen. Thus we derived expressions for the number of the random number calls we should make to generate all the probabilities for any sequence of data. The process can be described as follows: We first generate the sequences of all possible values, which the variables can take. These sequences are computed as a Distance Diminishing Sequence with the property that the distance between the subsequent sequences is unity. Once the sequences are generated we compute the probabilities associated with these sequences. These probabilities are computed in the same order as the sequences are derived. We also presented a formal algorithm to compute these sequences. To compute the joint probabilities of an arbitrary order, we showed how we could generate sequences of length k by using the probabilities already computed of order k-1.

Experimental results demonstrate that it is possible to build a joint probability for a distribution of N Boolean variables. This has been done for hypothetical distributions and for distributions suitable for the ALARM system, which has been extensively studied in the literature.

A summary of the results of this chapter is currently being compiled as a publication [OOM99B].

Chapter 5

Learning Polytrees from Sample Data

5.1 Motivation of the Problem

In the last chapters we described the algorithm for building a polytree structure if the exact distributions for the random variables were given. In this chapter, we shall proceed to consider the case when the joint or marginal probability distributions are not explicitly given. Rather, we consider the case when the user is only provided with samples made available from the application domain. Thus we now turn our attention to the more fundamental task of estimating the dependence polytree from the set of samples $\mathcal{S}$ instead of the probabilities and the joint distributions themselves. In fact, in practical applications most of the time probability distributions are not available, as only frequencies are obtained using the sample data. To achieve this we shall proceed in a systematic manner: the first step is to derive an estimate of optimal tree structure using the estimated information metric I_f or the chi-squared metric I_χ. The second step is to orient the arcs to obtain the polytree. The orientation process is based on a decision (an estimate) of independence between pairs of variables.

The fundamental assumption made by the polytree algorithm (see Chapter 3) is that in order to orient the tree the random variables which are sources or the parents of a node in the resulting polytree must be statistically independent. We shall now show that this assumption is not necessarily satisfied with real life data.

To view the problem in the overall picture we consider the following simple experiment, which involves independently tossing a pair of coins at each time instant. The possible outcomes are four namely, (H, H), (H, T), (T, H) and (T, T). Suppose the probability of the first coin yielding a head is x and the corresponding probability for the second coin is y. From these we see that the relevant random variables have the distributions shown below if they are statistically independent:

X=	H	T
$P_r(X= x)$	0.2	0.8

Y=	H	T
$P_r(Y= y)$	0.4	0.6

Y= \ X=	H	T
H	0.08	0.32
T	0.12	0.48

Table 5.1: The probabilities associated with X, Y and their joint distributions.

From the laws of elementary probability, we see that due to independence

$P_r(X= x, Y= y) = P_r(X= x).\ P_r(Y= y)$ for all x and y.

Conversely, if $P_r(X, Y)=P_r(X).\ P_r(Y)$ for all values of X and Y we can conclude that X and Y are independent.

Consider now a real life situation where, for example, we toss the coins 1000 times. In order to verify statistical independence we must have exactly the following frequencies shown in Table 5.2.

n_y \ n_x	H	T
H	80	320
T	120	480

Table 5.2: Table of frequencies required to verify independence of variables X and Y.

Consider now the probability of obtaining the frequencies of Table 5.2 required to justify the independence of the variables X and Y. This probability can be obtained from the multinomial distribution:

$$p_{1000}^{80\ 320\ 120\ 480} = \binom{1000}{80\ 320\ 120\ 480} (.08)^{80}(.320)^{320}(.12)^{120}(.48)^{480}$$

A straightforward computation shows that this probability is extremely small. We can therefore conclude that it is highly unlikely that we can verify independence exactly from any observed sample set.

In this chapter, we shall consider the problem of studying the dependence of two arbitrary variables and utilizing various "independence" decision rules obtained from the data. To achieve this we shall investigate the use of different metrics to quantify the dependence of two random variables if only samples are given, whence we can infer independence of the corresponding random variables and thereafter proceed to computing the polytree itself.

5.2 Building Polytrees from Samples

Throughout our presentation of the polytree algorithm we implicitly worked with the assumption that the "true" distribution of the random vector X was available to us. This knowledge enabled us to determine the optimal dependence tree and consequently the polytree. The situation changes if only a finite number of representative samples are presented to the procedure, which estimates the dependence tree. A method, which estimates the probabilities, will be used to yield an estimate of the dependence tree.

Let X^1, X^2,..., X^s be s independent samples of a finite discrete variable $\underline{X} = [X_1, X_2,..., X_N]^T$. In case of discrete variables, let

$$Freq_{UV}(X_i, X_j) = \frac{N_{UV}(X_i, X_j)}{\sum_{UV} N_{UV}(X_i, X_j)}$$

where $N_{UV}(X_i, X_j)$ is the number of samples with $X_i = U$ and the variable $X_j = V$.

Let $Freq_U(X_i) = \sum_V Freq_{UV}(X_i, X_j)$ where $Freq_{UV}(X_i, X_j) = Freq\ (X_i = U, X_j = V)$.

It is well known that $Freq_{UV}(X_i, X_j)$ is the Maximum Likelihood Estimate(MLE) of the probability P($X_i = U, X_j = V$). Using these estimates for the probabilities, the estimate of the information metric I will be the following:

$$I_{est} = \sum_{U,V} Freq_{UV}(X_i, X_j) \log \frac{Freq_{UV}(X_i, X_j)}{Freq_U(X_i) Freq_V(X_j)} . \quad (5.1)$$

We shall now present a procedure to build the polytree if we utilize the estimates I_f. For all pairs of variables X_i and X_j we compute the frequencies $Freq_{UV}(X_i, X_j)$ and $Freq_U(X_i)$ and using these frequencies we compute the quantity I_{est} defined in (5.1) above.

This quantity approximates the information metric EMIM used in Chapter 2 to build the optimal tree using Chow's algorithm, which is the skeleton of the polytree. Once the MWST is obtained, we proceed to obtain the orientation of the edges by using different strategies, which will be described, in the subsequent sections. However the problem encountered is that since we deal with finite samples, with a very high probability the quantity I_{est} will not equal zero which is what would be required for two variables to be independent.

Consider now the case when the vector $\underline{X} = [X_1, X_2, ..., X_N]^T$ is composed of continuous normal random variables $\{X_i\}$, and where $P(\underline{X})$ denotes the joint density function of the random vector $\underline{X}$. As in the case of discrete variables, to find the polytree for continuous variables, we worked with the assumption that the true covariance matrix of the random vector $\underline{X}$ was available. Using the parameters of the normal distribution of the random vector $\underline{X}$, we determined the optimal dependence tree using the information metric I_f. Now, if these parameters are unknown, and only sample data points are given to us, the first step would be to obtain the maximum likelihood estimate of the covariance matrix, and subsequently to derive the optimal tree using the estimated parameters. Again Chow *et al.*, [CL68] showed that in the case of tree dependence, the estimated tree is the maximum likelihood tree.

Let X^1, X^2,..., X^s be s statistically independent samples drawn from the underlying normal distribution. The MLEs of the parameters μ and Σ are given as:

$$\hat{\mu} := \frac{1}{s}\sum_{k=1}^{s} X^k \quad \text{, and ,}$$

$$\hat{\Sigma} = \frac{1}{s}\sum (X^k - \hat{\mu})(X^k - \hat{\mu})^T .$$

The MLE of the correlation coefficients between the components X_i and X_j are determined using the i^{th} and j^{th} components of the sample vectors. The MLEs of the covariance matrix can simply be obtained by estimating all pairwise correlation coefficients. Once the covariance matrix is known the procedure to obtain the tree can be followed to yield an estimate of the dependence tree. Since the MLEs of the correlation coefficients converge to their true values, one can expect that this procedure yields the true dependence tree. As proved in [VO92], the tree obtained by using the estimates $\hat{\Sigma}$ is the MLE of the dependence tree chosen from among all the possible trees, and furthermore, this estimate converges to the real unknown underlying tree as the number of samples increases.

Again, after obtaining the skeleton tree structure, the problem we encounter is one of evaluating the independence of every pair of variables to determine the orientation of the edges of the tree. Here too, we will consider various tests to quantify the independence of two random variables.

5.3 Independence of Random Variables

Probabilistically, the definition of independence between two variables X and Y uses the equality of functions P(X, Y) = P(X). P(Y) for all the values of X and Y. Conversely, to test independence of X and Y we have to test whether the joint distribution of X and Y equals the product of their marginal distributions. Independence in these conditions implies that all the joint moments can be partitioned as below:

$$E(X^iY^j) = \Sigma\Sigma\, X^iY^j\, P(X, Y) = \Sigma\Sigma\, X^i\, Y^j\, P(X)\, P(Y) = E(X^i)\, E(Y^j) \quad (\forall\, i, j > 0).$$

Since this is a very strong property, a first approximation to this can be obtained by only considering second central moments using the correlation coefficients. Let

$\mu_{11} = E[(X- \mu_x)(Y -\mu_y)]$, where

μ_x=E(X) and μ_y = E(Y) are first order moments. The correlation coefficient, ρ, of X and Y is a measure of the closeness to linearity of a relationship between the variables X and Y. Indeed, the correlation can be perceived as a "first-order approximation" to independence, and is defined as:

$$\rho(X, Y) = \mu_{11} / \sigma_x \sigma_y .$$

If ρ is equal to zero, the variables X and Y are uncorrelated. Since we are dealing with arbitrary distribution functions, the property of being uncorrelated does not necessarily imply their mutual independence. However, it is well known that for Gaussian distributions the uncorrelation of two variables implies their mutual independence. Observe though, that, in general, uncorrelatedness is a much weaker condition than independence.

To determine independence we shall now consider one scenario in which the number of samples or points is known. In such a case we obtain a decision rule to determine if two random variables are independent, by utilizing the well-known statistical Chi-Square hypothesis testing strategy with a specified confidence value.

5.3.1 Testing Independence for Discrete Variables

- **Using the Information Metric I_f**

If we are given probabilities, independence using the information metric implies the following:

$$I_f(X_i, X_j) = \sum_{X_i, X_j} P_t(X_i, X_j) \log \frac{P_t(X_i, X_j)}{P_t(X_i) P_t(X_j)} = 0.$$

- **Using the Correlation Coefficient**

The idea here is to use the correlation coefficient to quantify "the degree of dependence" between two discrete variables. First of all observe that it involves less computational effort than the information metric. Furthermore, it can be seen (empirically) that:

$[I_f(X, Y) \leq \alpha_1] \Leftrightarrow [\rho(X, Y) \leq \alpha_2]$ where α_1 and α_2 are arbitrarily small.

To demonstrate the above we present below the results obtained from a typical experiment. First we generated random samples for two independent variables X and Y and their corresponding ensemble average entropy and ensemble average absolute correlation were computed from the generated samples. The results are given in Figure 5.1. From the results we notice that as the number of samples increases, both the correlation and entropy curves converge to value zero for independent variables. Such results are typical and are as expected from the Central Limit Theorem where the sequence of Bernoulli trials converges to a normal random variable in which case the uncorrelatedness is equivalent to independence.

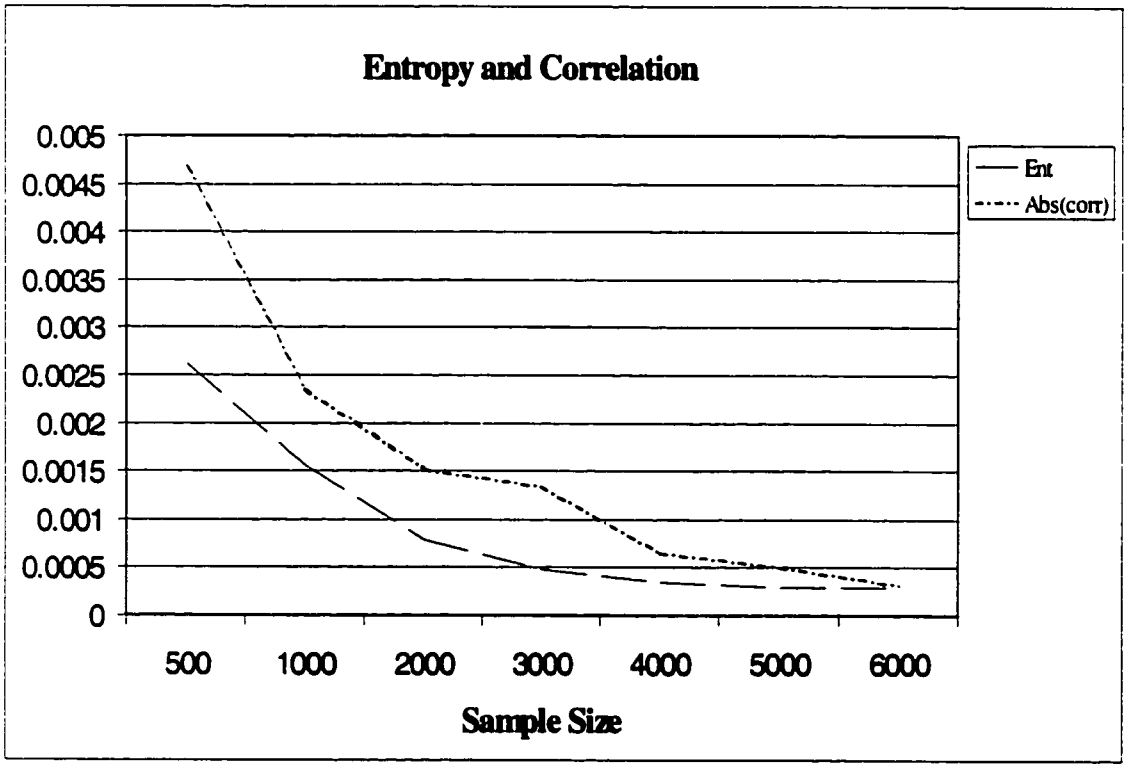


Figure 5.1: The behavior of the ensemble average absolute correlation and the ensemble average entropy for two independent variables as a function of the sample size.

From the above graph (which, is typical for independent random variables) we see that the correlation being zero is closely related to the entropy being zero, and consequently

we can use the former to serve as an indicator to decide whether the random variables are "independent", which, is a crucial step in the polytree algorithm.

- **Using the Chi-squared Metric**

If we use the Chi-Squared metric for testing the independence of two random variables we will evaluate the quantity:

$$I_\chi(X_i, X_j) = \sum_{X_i, X_j} \frac{(P(X_i, X_j) - P(X_i)P(X_j))^2}{P(X_i)P(X_j)}$$ and require it to be equal to zero.

We know that:

(i) $I_\chi(X_i, X_j) \geq 0$

(ii) $I_\chi(X_i, X_j) = 0$ if and only if $P(X_i, X_j) = P(X_i)\, P(X_j)$.

Independence using these conditions is not an interesting test, because it implies that we have to test if $I_\chi(X_i, X_j) = 0$ (i.e., $P(X_i, X_j) = P(X_i)\, P(X_j)$) which poses exactly the same problem that we have already addressed earlier, and which usually cannot be satisfied with real life data. Consequently, we will have to use additional information contained in the number of samples to derive χ^2 tests, which implicitly use hypothesis-testing strategies. This will be discussed in a later sub-section.

5.3.2 Independence for Continuous Normal Variables

Having described the use of the I_f metric for the discrete random vector case, we shall now proceed to consider the case when the vectors are continuous. The question is a little less complicated when we are studying the independence of two continuous normal variables because in this case, as we have already mentioned, independence and uncorrelation are equivalent. Consequently, since the correlation coefficient between two random variables is much easier to compute than the information metric, the former will be used to determine the orientation of edges of the optimal tree.

In the case of normal random variables the marginal density of the random variable X_i has the usual uni-variate normal density $P(X_i)$,

$$P(X_i) = N(\mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\{-\frac{1}{2}\left(\frac{X_i - \mu_i}{\sigma_i}\right)^2\}$$

where σ_i is the square root of the diagonal entry σ_{ii} of the covariance matrix Σ.

Also the joint distribution of the pair of random variables X_i and X_j has the density $P(X_i, X_j)$ given by:

$$P(X_i, X_j) = \frac{1}{2\pi\sigma_i\sigma_j(1-\rho_{ij}^2)^{1/2}} \exp\{-q(X_i, X_j)\}, \text{ where}$$

$$q(X_i, X_j) = \frac{1}{2(1-\rho_{ij}^2)}\left[(\frac{X_i - \mu_i}{\sigma_i})^2 + (\frac{X_j - \mu_j}{\sigma_j})^2 - 2\rho_{ij}(\frac{X_i - \mu_i}{\sigma_i})(\frac{X_j - \mu_j}{\sigma_j})\right]$$

and $\rho_{ij} = \sigma_{ij} / \{\sigma_i \sigma_j\}$ is the correlation coefficient between X_i and X_j. As it is well known, if the correlation coefficient is equal to zero the joint distribution $P(X_i, X_j)$ can be factored and written as follows:

$$P(X_i, X_j) = \frac{1}{2\pi\sigma_i\sigma_j} \exp\{-q(X_i, X_j)\} \text{ where } q(X_i, X_j) = \frac{1}{2}\left[(\frac{X_i - \mu_i}{\sigma_i})^2 + (\frac{X_j - \mu_j}{\sigma_j})^2\right]$$

which is the product of the marginal densities of the random variables X_i and X_j. Thus if the correlation coefficient is equal to zero the variables are independent, and vice versa, implying that for jointly normal random variables independence and uncorrelation are equivalent.

• Using the Information Metric I_f to Determine Independence

When the variables are continuous the information metric is:

$$I_f(X_i, X_j) = E\left[\log\frac{P(X_i, X_j)}{P(X_i)P(X_j)}\right].$$

Unlike the discrete case (in which the evaluation of the information metric between two variables depends only on the discrete probabilities of the joint events), in the continuous case, the value of $I_f(X_i, X_j)$ is derived by computing the corresponding integral over all values of X_i and X_j. For jointly normal pairs of random variables X_i and X_j, it can be seen [VO92] that the information metric between these variables is given by:

$$I_f(X_i, X_j) = E\left[\log\frac{P(X_i, X_j)}{P(X_i)P(X_j)}\right] = -\frac{1}{2}\log(1 - \rho_{ij}^2) \text{ where}$$

$\rho_{ij} = \sigma_{ij}/\{\sigma_i\sigma_j\}$ is the correlation coefficient between X_i and X_j.

- **Using the Correlation Coefficient**

In this case, we don't need any approximation between the information metric and the correlation coefficient because independence of two variables implies their mutual uncorrelation. We thus invoke the equivalence between the quantities $I_f(X_i, X_j)$ and ρ_{ij} and thus:

$$(I_f(X_i, X_j) = 0) \Leftrightarrow (\rho_{ij} = 0)$$

Therefore, if ρ_{ij} is arbitrarily small, it is easy to see that this is an arbitrarily good approximation of the random variables being mutually independent.

- **Using the Chi-Square Hypothesis Testing Strategy**

Having described the use of the correlation coefficient to determine the independence of two random variables for discrete and continuous distributions, let us now turn our attention to the problem of using the well known Chi-square hypothesis testing strategy to find whether two random variables are independent or not.

The need to decide whether a particular claim or statement is correct is often the motivation for a statistical test of a hypothesis. The decision to reject or accept a certain hypothesis is based on a statistic called a *test statistic* computed from sample data. The steps that we are going to describe apply to all tests of hypotheses except that the form of the test statistic and the rejection region change for different applications. In spite of these the sequence of steps is always the same. The steps are:

Step 1: Set up the null hypothesis H_0 and the alternative hypothesis H_1. If the purpose of the hypothesis test is to test whether the variables are independent the "independence hypothesis" is always stated as H_0, and the "dependence hypothesis" is stated as H_1.

Step 2: Define the test statistic. This is evaluated using the sample data to determine if the data is compatible with the null hypothesis.

Step 3: Define a rejection region, having determined a value for α, the significance level. In this region the value of the test statistic will result in rejecting H_0.

Step 4: Calculate the value of the test statistic, and carry out the test. Using the test state the decision of whether to reject H_0 or to reject H_1.

Step 5: Give a conclusion in the terms of the original problem or question. This statement should summarize the results of the analysis.
An example should clarify the above process.

Example 5.1: Let X and Y be two random binary variables. Given a table of observed frequencies, the problem is to determine whether X and Y are related or independent. For this, we use hypothesis-testing strategy with level of significance equal to 0.05.

H_0: X and Y are mutually independent.

H_1: There is dependence between X and Y.

Let us assume that the observed frequencies yield us the following table:

	X=true	X=false	
Y=true	20	60	→ 80
Y=false	40	150	→ 190
	↓ 60	↓ 210	

Table 5.3: The Observed Frequencies of the random variables X and Y.

To find the expected frequencies, if there is independence between X and Y, the expected frequency in a cell is the product of the marginal totals in the row and column to which the cell belongs divided by the overall total, assuming H_0 true (P(X, Y) = P(X)*P(Y)).

The expected frequency table is given in Table 5.4. The final χ^2 statistic is given in Table 5.5 where o represents the observed frequency and e represents the estimated frequency.

	X=true	X=false
Y=true	$\frac{60(80)}{270} = 17.7$	$\frac{210(80)}{270} = 62.2$
Y=false	$\frac{60(190)}{270} = 42.2$	$\frac{210(190)}{270} = 147.7$

Table 5.4: The Expected Frequencies of the Random Variables X and Y.

X	Y	o	e	o-e	$(o-e)^2$	$(o-e)^2/e$
true	true	20	17.7	2.3	5.29	0.298
false	false	150	147.7	2.3	5.29	0.035
true	false	40	42.2	-2.2	4.84	0.114
false	true	60	62.2	-2.2	4.84	0.077

Table 5.5: χ^2 Statistic based on the Sample Data

In a 2 by 2 dimension table, the degree of freedom D = (2-1)(2-1) =1, and from the table of the Chi-square distribution, for D =1 and $\alpha = 0.05$ yields $\chi_2 = 3.84$. From Table 5.5, we see that $\sum \frac{(o-e)2}{e} = 0.526$ which is less than 3.84, and thus the hypothesis H_0 (that X and Y are independent) is accepted. We can thus assert that X and Y are independent.

Thus, if the number of sample points is known [11]we shall use the χ^2 hypothesis test to determine the independence of the nodes, whence the polytree can be computed. This is explained in the following section.

5.4 Polytree Sample Algorithms

Using different tests for stating independence or dependence of two random variables, we can proceed to the development of the various algorithms for the case when the samples data is available to the procedure which estimates the optimal dependence tree and polytrees. The first algorithm is described for the case of discrete variables, and the next one will be for the continuous case. Observe that the algorithm is analogous to the discrete case except for the actual tests. The Polytree algorithm invokes different procedures (for example, a procedure to build the MWST using the estimated edges weights), a procedure to orient the tree to yield the corresponding polytree using any one of the tests for independence of parents of nodes, which invokes a procedure to compute the independence of variables.

[11] It is possible that the number of samples is not known, but only estimates of the probabilities are given. This is the scenario when correlation tests are done to determine independence.

Algorithm Polytree-Discrete-Samples
Input: The set $\mathcal{S}$ of samples X^1, X^2,..., X^s.
Ouput: The estimate of the polytree
Method
Begin
For (i =1 to N and j < i) **Do**

Estimate $I_{est} = \sum_{U,V} Freq_{UV}(X_i, X_j) \log \frac{Freq_{UV}(X_i, X_j)}{Freq_U(X_i) Freq_V(X_j)}$ **or**

Estimate $I_\chi(X_i, X_j) = \sum_{X_i, X_j} \frac{(Freq_{UV}(X_i, X_j) - Freq_U(X_i) Freq_V(X_j))^2}{Freq_U(X_i) Freq_V(X_j)}$

EndFor
Repeat Until A Spanning Tree T is completed
Include the branch with largest weight unless a cycle is obtained.
If cycle is obtained **Then**
Discard edge.
EndIf
EndRepeat
Call Procedure Polytree-Depth-First-Search
End Algorithm Polytree-Discrete-Samples

The algorithm Polytree-Depth-First-Search is already described in Chapter 3. Its purpose is to orient the tree T. It invokes a procedure called Processing to allow the orientation of the tree recursively, which uses a test for independence of variables. In algorithm Polytree-Discrete-Samples, the independence test is replaced by the procedure called Independent and described below. This procedure takes as input two nodes and determines whether they are independent or not using different tests with the sample data.

Procedure Independent(node1, node2)
Input: node1, node2; error level ε and confidence level α.
Output: True if node1 and node2 are independent.
Method:
Begin
    If (number of points N is known) **Then**
        Call Chi-Square Hypothesis Testing (N, α)
    ElseIf (correlation (node1, node2) < ε) **Then**
        Return True
    Else
        Return False
    EndIf
End Procedure Independent

Since we have now described the polytree algorithm for discrete case, we will proceed to give the algorithm for Normal variables. The procedure Polytree-Depth-First-Search for orienting the tree is the same as in the discrete case but for independence tests we use the correlation coefficient since uncorrelated variables are equivalent to independent variables in this case.

5.5 Algorithm Polytree-Continuous

This algorithm shares some procedures with the discrete case. In fact the orientation of the MWST is done in the same manner as in the discrete case and the traversing of the tree to obtain the polytree is also done in the same way as in the discrete case.

Algorithm Polytree-Continuous-Samples
Input: The set $\mathcal{S}$ of samples X^1, X^2,.., X^s.
Ouput: The estimate of the polytree
Method:
Begin

Compute the Maximum Likelihood Estimates (MLE) of the distribution of X.

$$\hat{\mu} := \frac{1}{s}\sum_{k=1}^{s} X^k \text{ and}$$

$$\hat{\Sigma} = \frac{1}{s}\sum (X^k - \hat{\mu})(X^k - \hat{\mu})^T .$$

Compute R:= $[\rho_{ij}]$, the matrix of correlation coefficients from $\hat{\Sigma}$.

For (i=1 to N and j<i) **Do**

Compute the measure $I_f(X_i, X_j) = E\left[\log \frac{Freq_{UV}(X_i, X_j)}{Freq_U(X_i)Freq_V(X_j)}\right]$ **OR**

Compute the measure $I_\chi(X_i, X_j) = \frac{\rho_{ij}^2}{1-\rho_{ij}^2}$.

EndFor
Repeat Until A Spanning Tree T is completed
Include the branch with largest weight unless a cycle is obtained.
If cycle is obtained **Then**
Discard edge
EndFor
EndRepeat
Call Polytree-Depth-First-Search (T) to obtain the polytree

End Algorithm Polytree-Continuous-Samples

These algorithms have been implemented in the 'C' language, and in order to test them we have implemented the sample generation algorithms already described in Chapter 4. In fact, we generated data from either tree structures or polytree structures and we used these pieces of data as input to the polytree algorithms. We also used the generated samples to verify the quality of the approximation of the independence of the variables. The following section describes all these results.

5.6 Experimental Results

To verify the strength of our techniques, in this set of experiments, instead of using the probability distributions, we used an underlying tree or polytree to generate the samples. The estimates of the first and second order marginals were updated with every sample. The Chow tree was computed as the samples are generated. First we ran the algorithms on tree structures. Both Pearl's method and our method for generating random samples were used depending on whether the conditionals were available or the first order probabilities were given. The following sections report the results we have obtained in some typical cases. Of course, it is not possible to report all the results we have obtained[12], but the results reported are representative. We also notice that the branching factor and the number of nodes of the trees represent the parameters that influence the running time of the algorithm.

5.6.1 Inferring Known Tree Structures

If the underlying tree was given we generated samples obeying that tree and from the generated samples we built the Chow tree using the estimated information measure between the variables as edge weights. Our experimental results clearly show the number of samples needed to reproduce the underlying tree.

5.6.1.1 Using Algorithms for First Order Probabilities

We generated samples following the given structure and from these samples we built the Chow tree. Two example trees are the trees given in Figures 5.2a and 5.2b. The learning methods gave the results shown in Figure 5.3.

[12] In most cases the underlying structure was specified manually. No systematic procedure was used to specify the parameters of the underlying polytrees. They were subjectively created. In some cases we relied on the data used by other researchers [Die 97].

Example 5.1:

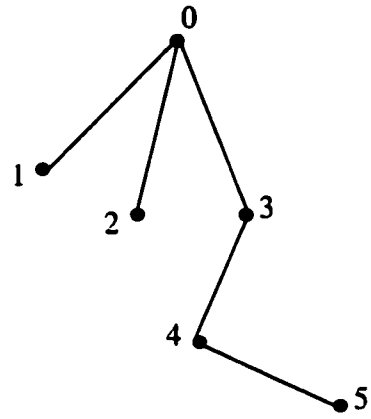


Figure 5.2a: Given Tree Structure (a).

Example 5.2:

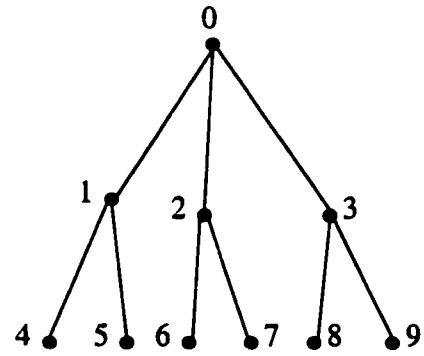


Figure 5.2b: Given Tree Structure (b)

Figure 5.3 summarizes the results of the scheme using the trees given in Examples 5.1 and 5.2. The term mismatches is used to indicate the number of edges in the underlying tree that is not recovered. For Example 5.1, the learning algorithms used only 300 samples to recover the tree given in Figure 5.2a. We notice that as the number of nodes increases the number of samples required to recover the underlying tree also increases.

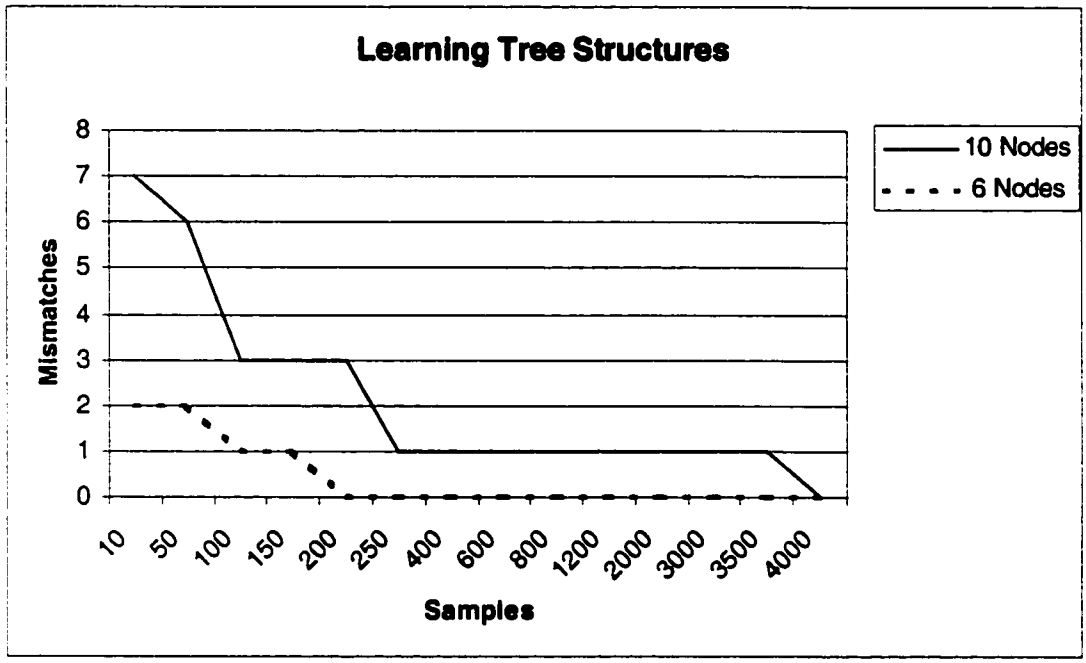


Figure 5.3. Graph demonstrating the average rate of learning for tree structures for two typical trees.

5.6.1.2 Data Generation: Pearl's Algorithm

In this case we assumed that all the conditional probabilities were given and we generated samples for the tree structures given in Figures 5.2a and 5.2b. We derived the samples and thus computed the tree. The results obtained are the same as those obtained using the sample generation method when first order probabilities are given. The only difference between Pearl's method and this method is that it takes more time for the later algorithm to build the conditional probabilities. Once these conditionals are built the same number of samples are needed to recover the underlying trees for both methods.

5.6.2 Inferring Known Polytree Structures

To test the polytree learning algorithms presented in this chapter, we proceeded as follows. If the underlying polytree is known we generated samples following that structure as per the technique discussed in Chapter 4. Once the samples were generated,

we built the Chow tree. To orient the obtained tree, in the first experiment we first assumed that the results of the independence tests were given. For the second experiment we computed them from the samples generated using one of the methods discussed earlier, and compared the derived polytree to the "true" underlying polytree. About ten examples[13] were used to test the algorithms, but in the interest of brevity we give the results of two such polytrees. We assumed the polytrees given in Figures 5.4a and 5.4b as the underlying structures for generating samples.

- **Known Independence between Variables**

Example 5.3:

Using the underlying polytree of Figure 5.4a, we assume that nodes 0 and 1 are independent.

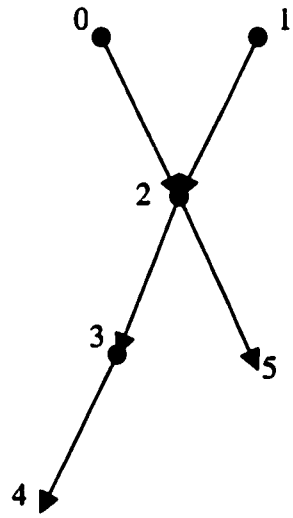


Figure 5.4a: Given Polytree with 6 nodes

[13] See footnote on Section 5.6 in page 122.

Example 5.4:

In this example, we considered structure with many independent variables. Nodes 0, 1 and 2 are assumed independent.

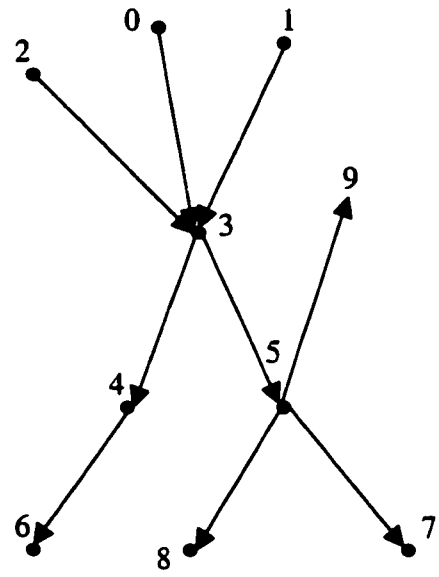


Figure 5.4b: Given Polytree with 10 nodes

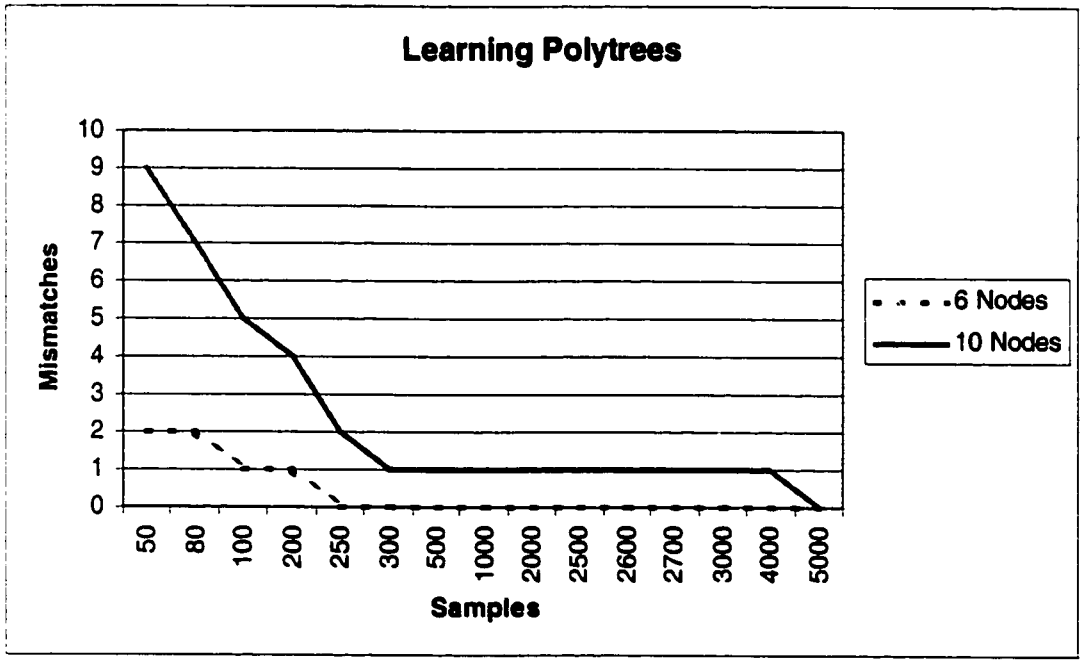


Figure 5.5: Graph describing the rate of learning polytrees for two typical polytrees.

The results of running the polytree learning algorithms on the polytrees given in Figures 5.4a and 5.4b is shown in Figure 5.5 when the independence between variables is given. It was observed that if we generated samples from an underlying tree and tried to recover the tree, it took less samples to recover the tree than if we generated samples from a polytree and tried to recover that polytree. We also ran the experiments using the Chi-square metric and we got the same trees and polytrees as with the information measure. Indeed, the results are identical.

- **Using the Estimated Probabilities for Independence**

In the case where the independence information is not available, we derived the independence information from the estimated probabilities computed from the generated samples. The independence tests were obtained by either computing the statistical equality for independent variables or using the correlation between them. The results obtained using the statistical hypothesis testing methods were categorically poor. Indeed, we could not achieve satisfactory learning even after 500,000 samples for the polytrees given in Figures 5.4a and 5.4b.

- **Using the correlation coefficient**

The correlation coefficient for independent variables decreases as the number of samples increases. We computed the correlation coefficient for independent nodes for the two polytrees given in Figures 5.4a and 5.4b. The results are shown for an ensemble average of 10 runs in the following tables, and the graphs are given in Figures 5.6 and 5.7.

Number of Nodes	Number of Samples	Correlation (0,1).
6	500	0.035
6	5000	0.005
6	50000	0.001

Table 5.6: The average correlation coefficient for independent variables for the polytree given in Figure 5.4a.

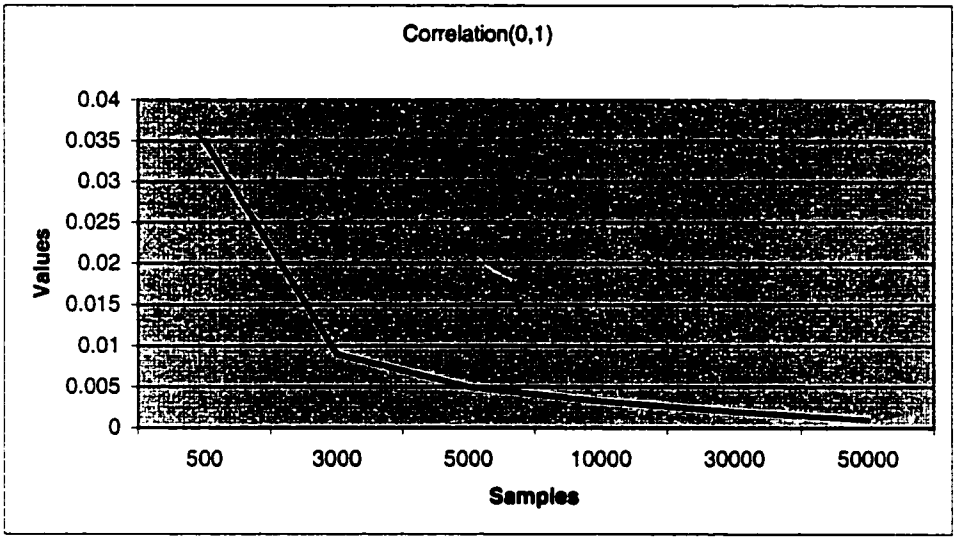


Figure 5.6: Graph describing the average rate of learning independence for nodes 0 and 1 in polytree given in Figure 5.4a.

Number of Nodes	Number of Samples	Correlation (0,1)	Correlation (0,2)
10	5000	0.035	0.004
10	50000	0.005	0.002
10	500000	0.001	0.001

Table 5.7: The correlation coefficient for independent variables for the polytree given in Figure 5.4b.

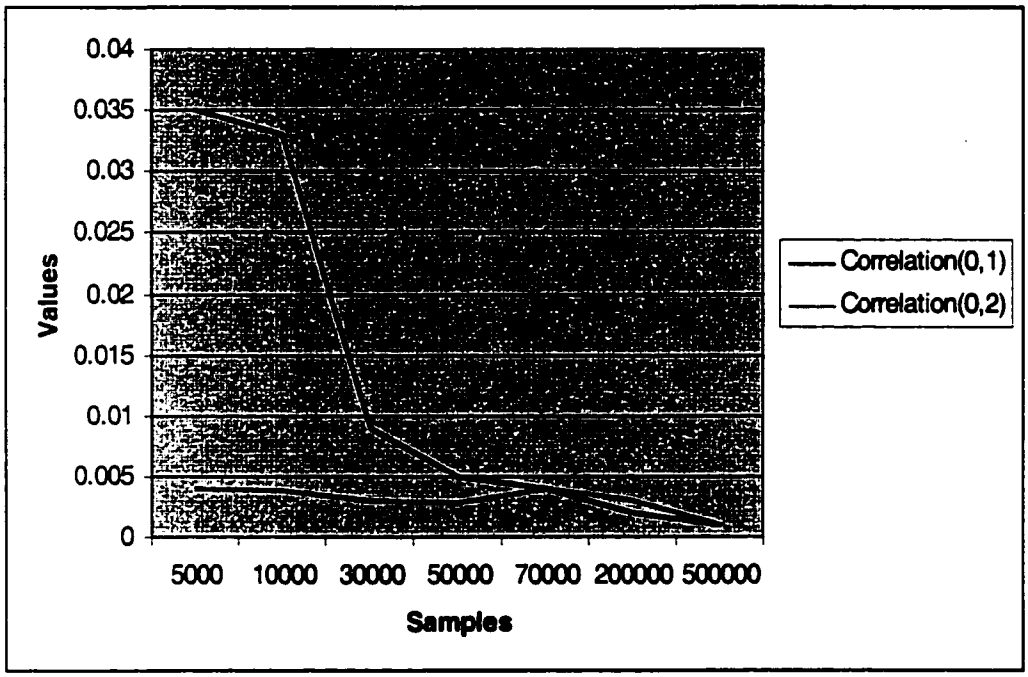


Figure 5.7: Graph describing the rate of learning independence for nodes 0 and 1 in polytree given in Figure 5.4b.

5.7 Real –Life Application I (Alarm Network)

In this section we describe an experiment in which we used the Alarm belief network which was created by Ingo Beinlich in [BSCC89] in 1989. As an anesthetist, Beinlich constructed the Alarm belief network structure and filled in all the corresponding probability tables for the nodes to model anesthesia problems. Later, in 1991, the Alarm database, which contains 20,000 cases, was generated by Edward Herskovits in connection with his Doctoral dissertation given in [Her91]. This database of cases is generated using the technique developed by Henrion for belief networks [Hen88]. The Alarm database is widely used in the machine learning community especially for methods related to the induction of Bayesian belief networks from data. The best reference for this is the work by Cooper *et al.*, in [CH92] for the induction of probabilistic networks from data. This work was described in Chapter 2. Another related work, which used the Alarm database, is given in [HGC95]. Experimental results are reported in these references showing the structure that these methods recover and the time required for them to recover the structure of the underlying network.

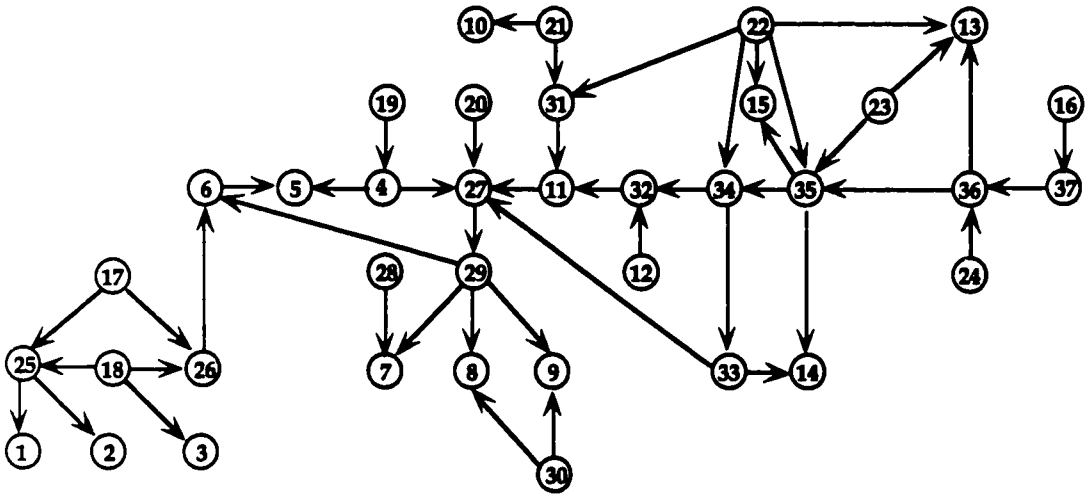
The Alarm database contains 20,000 cases each describing an instance of the variable vector. This variable vector is composed of 37 variables forming the nodes of the Alarm network as it is shown in the figure below taken integrally from [Her91]. These variables constitute the nodes of the Alarm network and each node has from two to four possible values. Also the Alarm network contains a total of 46 arcs and 37 nodes. For simplicity the node names are replaced with numbers and the corresponding names are given in Figure 5.8.

We used our algorithms and ran them on the 20,000 cases. First we computed the information measures for the edge weights of the complete graph and build the Chow tree (the maximum weight-spanning tree) which obviously includes exactly 36 edges since we have 37 nodes in total. This skeletal tree is given in Figure 5.7. The Chow tree was obtained in about 3 minutes when using the 20,000 samples. We proceeded to orient the obtained tree to get the polytree network. In this specific case we know that the distribution we used in not that of polytree based dependence since the samples are derived from the Alarm network which is given in Figure 5.6. The objective is to induce a

polytree that "best describes" the probability distribution using the training data. In practice, as it is described in Chapter 2, this optimization process is implemented using heuristic search techniques to find the best network, which fits the training data. In our case we are approximating the distribution by a polytree-based dependence and this polytree approximation represents the bias for the algorithms we have used. The polytree structure, which is obtained from the samples, does not include all the edges of the Alarm network since the inclusion of certain edges will violate the definition of the polytree. One of the benefits of the reported algorithms used is that they are very efficient. Furthermore, our method does not require any ordering on the nodes of the network.

As opposed to the above, in [CH92], the authors reported that the K2 algorithm constructed a network identical to the Alarm network except that the arc from node 12 to node 32 was missing and the arc from node 15 to node 34 was added. The authors of [CH92] ran the K2 algorithm on only 10,000 cases and the results are obtained in 16 minutes and 38 seconds. Also as mentioned by the authors of [CH92] themselves, the performance of the K2 algorithm is very sensitive to the ordering on the nodes in the Alarm network.

It is also pertinent to mention that recently in [Mei99], the author of [Mei99] developed an accelerated Chow and Liu algorithm. This algorithm takes advantage of sparse data to improve on the time and memory requirements of the skeletal tree learning algorithm. In the context of the present research, this result is encouraging because this will speed up our polytree learning algorithm especially for high dimensional domains as in the case of our application described in Chapter 6.



The ALARM belief network

KEY:

1 - central venous pressure
2 - pulmonary capillary wedge pressure
3 - history of left ventricular failure
4 - total peripheral resistance
5 - blood pressure
6 - cardiac output
7 - heart rate obtained from blood pressure monitor
8 - heart rate obtained from electrocardiogram
9 - heart rate obtained from oximeter
10 - pulmonary artery pressure
11 - arterial-blood oxygen saturation
12 - fraction of oxygen in inspired gas
13 - ventilation pressure
14 - carbon-dioxide content of expired gas
15 - minute volume, measured
16 - minute volume, calculated
17 - hypovolemia
18 - left-ventricular failure
19 - anaphylaxis
20 - insufficient anesthesia or analgesia
21 - pulmonary embolus
22 - intubation status
23 - kinked ventilation tube
24 - disconnected ventilation tube
25 - left-ventricular end-diastolic volume
26 - stroke volume
27 - catecholamine level
28 - error in heart rate reading due to low cardiac output
29 - true heart rate
30 - error in heart rate reading due to electrocautery device
31 - shunt
32 - pulmonary-artery oxygen saturation
33 - arterial carbon-dioxide content
34 - alveolar ventilation
35 - pulmonary ventilation
36 - ventilation measured at endotracheal tube
37 - minute ventilation measured at the ventilator

Figure 5.8: The Alarm Network taken from [Her91].

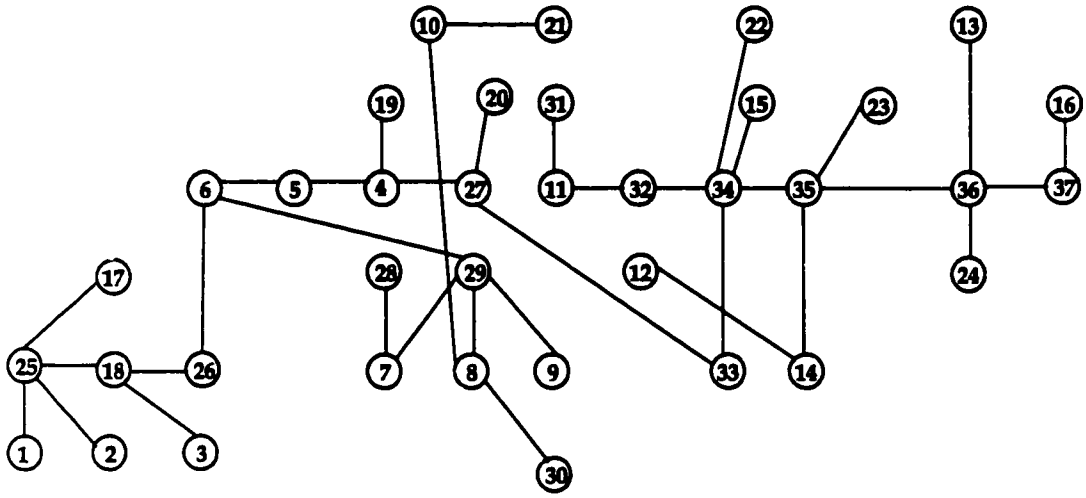


Figure 5.9: The Alarm Tree obtained by the Chow algorithm

The reader will observe that in Figure 5.9 some edges are missing with respect to the networking of Figure 5.8. In fact, because of the bias on the polytree structure these edges will not allow us to have a singly connected network. This is, in fact, quite reasonable, since the underlying Alarm network leads to a multiply connected network. Also, the edges 10-8, 12-14 and 15-34 are added by the algorithm to the structure because they have the highest information measure after removing the edges which form the cycles in the structure of the Alarm network.

In order to orient the Alarm tree we used the correlation between variables to obtain results for independence tests. The correlation coefficients were computed using the given 20000 sample cases of the Alarm network. We also approximated independent variables using a given threshold for the above correlation. After running the DFS orienting algorithm, only three edges remained non-oriented because they were not contained in any causal basin. The Alarm polytree is given in Figure 5.10 below. The power of our scheme is clear.

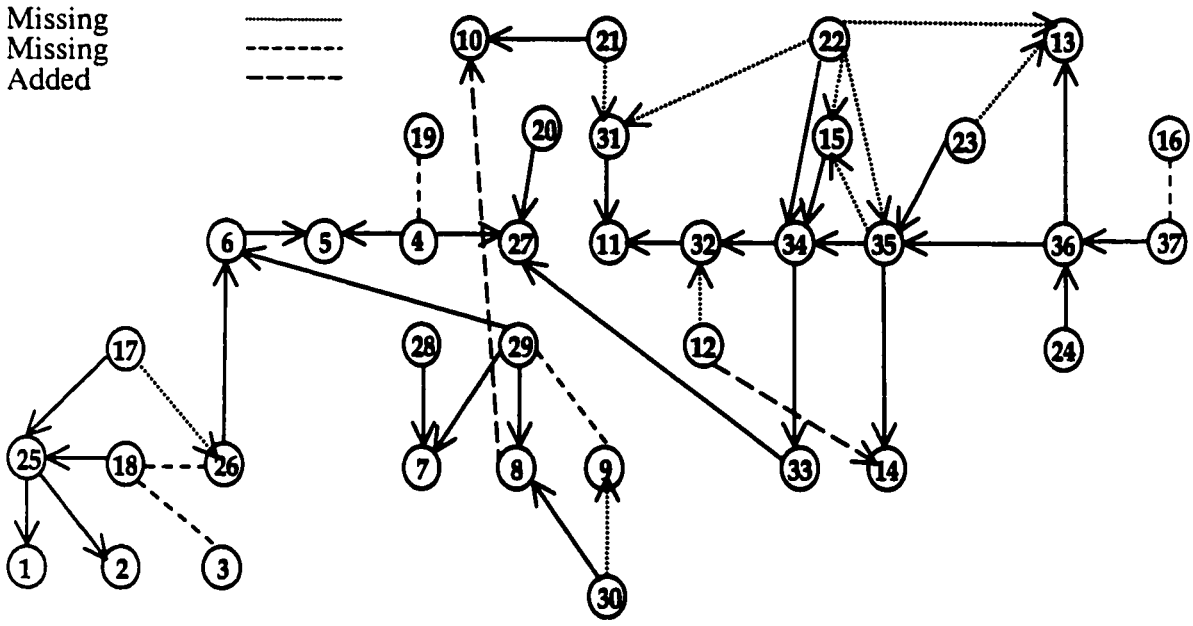


Figure 5.10: The Alarm Polytree after the DFS Algorithm.

The following graph shows the convergence rate of the number of matches as a function of the number of cases processed.

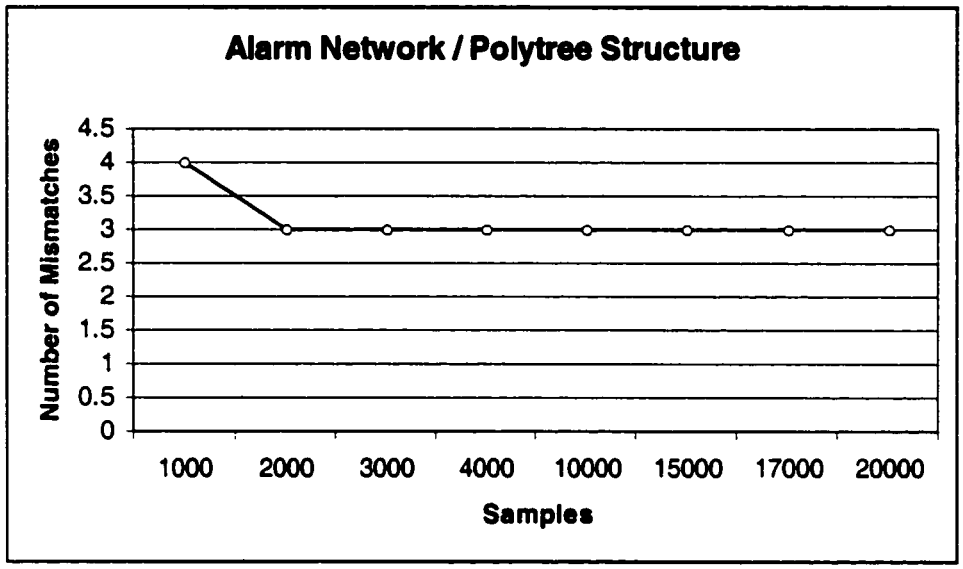


Figure 5.11: The Number of Mismatches between the Alarm Network and the Polytree obtained with the DFS Algorithm.

5.8 Summary

In this chapter, we have considered the problem of orienting the optimal skeletal tree structure obtained by the Chow algorithm using the estimated independence. The orientation process is based on a decision of independence between pairs of variables. We have considered various methods to infer independence including methods that make use of the information metric, a Chi-Square hypothesis testing scheme and the use of the correlation coefficient. We showed that for independent variables, the correlation is very small, and this allows us to infer independence of two variables, which, in turn, permits the orientation of the polytree structure. The main advantage with the correlation coefficient is that it is computationally less intensive than the information metric. We also implemented all the algorithms already developed and tested them on the samples generated from different structures. To demonstrate the applicability of the techniques, we also considered a real-life application, which is the Alarm network [Her91]. All the

required algorithms have been both implemented and tested for multi-feature variables since it is the case for the Alarm data. In all the cases, the results obtained were conclusive and fairly impressive. A more objective comparison of the polytree distribution and the K2 distribution could be done using a Kullback-Leibler [KL51] distance measure, but this is a problem for future study.

A summary of the results of this chapter is currently being compiled as a publication [OOM99C].

Chapter 6

Learning from the Use of Distributed Databases

6.1 Motivation of the Problem

A database is a logically coherent collection of data items related to some real-life phenomena or application. A DataBase Management System (DBMS) is a set of programs or software packages that facilitates the process of constructing and manipulating databases for users' applications.

A distributed database is a collection of multiple logically interrelated databases distributed over a computer network [OV91]. In a distributed environment, the data items are placed across the sites and are either partitioned or duplicated. In the partitioned design the database is divided into a number of disjoint partitions each of which is stored at a different site. Duplicated designs involve storing the entire database at each site or storing duplicates of some partitions of the database in several sites.

A relational database is one in which the database is represented as a collection of relations where a relation is described as a table of values, and where each row in the table represents a collection of related data values [EN94]. In a relational system, data is perceived by the user as tables only [Dat86]. Data manipulation languages (also called query languages) developed for the relational model are divided into two categories, which are: relational algebra-based languages, and relational calculus-based languages. The difference between them is based on the formulation of the query. When specifying a relational algebra query the user must specify *how* (i.e., in what order) the query operations have to be executed. The relational calculus-like Structured Query Language (SQL) is a declarative language, and so the user only specifies which relationships should be satisfied in the final result. A query expressed in a high–level query language such as SQL is typically transformed (converted) into an equivalent lower level query. Such a

given query usually has many possible correct transformations, implying that there are often many execution strategies. The process of choosing a suitable execution strategy for processing a query is known as query optimization [OV91] and is one of the most studied problem in query processing. The problem is to optimize the strategy for executing a given query under the constraint of a given metric and goal. For example, one possible goal in query optimization could be to minimize costs using a cost metric based on access performance and communication costs. Furthermore, the purpose of query optimization is to find an execution strategy for the query, which is close to optimal, for indeed, the problem of determining the optimal solution, is computationally intractable [IK84].

In a distributed system, besides solving the question of ordering (permuting) relational algebra operations, the query optimizer must also select the best sites at which the data has to be processed so as to minimize, for example, the communication costs. This, of course, increases the solution space from which to choose the distributed execution strategy making distributed query optimization far more difficult than its centralized counterpart. Most distributed systems that use a database management system consider communication costs as the most significant factor especially for Wide Area Networks (WANs). This is further clarified by the details presented in Table 6.1 below.

The typical distributed system in a database consists of a client-side application, and a distributed database. The components of the database may be either local, i.e., at the client site, or remote, in which case it is situated on a network node that is different from that of the client.

In a client-server application, the client application communicates using a binding[14] to a query processor on a remote database server. The client application invokes a binding service again for each access made to the data, which is returned by the query. The client application relies directly on the binding service.

[14] Bindings are defined as associations between object and interfaces. The binding specifies the role that each object has to play at the interface.

In our research, we shall approach query optimization in a way that is for the most part, novel to the field of distributed systems. The objective of our approach is to lessen the communication cost by avoiding repeated queries. It is novel to the distributed database research in the sense that we do not know of any work in caching of distributed databases which uses a machine learning approach. In general, query optimization methods for distributed database systems mostly depend on communication cost. But in order to minimize that component, most of the research algorithms consider the following factors: transportation cost, order of join, access method, method of join, and join site, and that of finding the best way for executing the query. Instead of choosing the best execution strategy for a query, we shall attempt to minimize the communication cost by minimizing the number of queries made to **remote** databases since the communication time is the most significant component in processing a request in a distributed environment. This optimization is done by learning the data workflow of the application that is generating the queries. The example shown in Table 6.1 illustrates the importance of communication time when accessing a distributed database and on fetching data.

Example 6.1: (Adapted from [Gov 97])

Assuming a typical 30 Kbytes query result on a Windows NT client using a 28.8 Kbps modem, a propagation delay of 0.25 secs, a Windows NT server, and with negligible query processing time, the response time for the query can be estimated as follows:

Time	Component	Network	Server
start	Generate request		
0.001 s	Queued for sending		
0.003 s	Transmission time		
0.25 s		Propagation delay	
0.001 s			queued for receiving
~ 0			received by listener
~ 0			perform action(s)
0.01 s			generate response
0.01 s			queued for sending
8.53 s			transmission time for 30 Kbytes
0.25 s		Propagation delay	
0.01 s	Queued for receiving		
end	Received by listener		
9.11 s	Response time		

Table 6.1: Client Query Execution Time for a Distributed Database

Note that approximately 90 % (i.e., 8.53/9.11) of the time delay is due to the time required to transmit the remote data. A large portion of this delay is avoidable, as described below.

Our approach to solve this problem is to reduce the number of accesses to remote database(s) to a minimum. Our intention is to make the system perform a trace of queries, which are stored in remote relational databases. We will then use these traces to build rules to allow decisions on caching at run time at the client server so as to avoid re-fetching of the same data, and also to anticipate the fetching of new data from what has been learned in terms of causal dependencies among queries. The pre-fetch of data, based on the anticipated need, will provide an improved performance when the data is to be subsequently accessed.

First of all, observe that we avoid the re-fetching of the same data. Indeed, the queries and their answer sets are cached in local memory of the client so that repeated access to the same query has *no* communication overhead, thus drastically increasing the performance. Secondly, we perform anticipated caching by learning the patterns or the workflow of the data, thus specifically utilizing the causality to predict future data usage. For instance, if a given query

Q1 = Select (col X1, table Y1, where X1=1)

Always precedes a subsequent query

Q2 = Select (col X2, table Y2, where X2=4),

it is easy to see that the overall operation can be optimized if when fetching data for query **Q1** we also do anticipated caching by fetching the data required for query **Q2**.

6.2 Related Work: *NetCache*

A partial solution to the problem of access to distributed databases has been addressed by the *NetCache* [KSL95] software from King Systems Limited (KSL). We expect our system to be a continuation and improvement of their software. The intention is that our software will be able to make **decisions** on which queries are to be cached and which queries to be discarded instead of caching all the queries. This process is achieved by learning the workflow of the application rather than comparing every new request to the previously cached answers for decisions regarding caching.

NetCache uses a very simple approach to cache management. It is a network caching software package, which stores all the answers to previously resolved queries up to the capacity of the memory, and then, whenever a new query is presented, the system compares it to the ones which are already cached in local memory and returns the answer to the user. Each query/answer set combination is assigned a unique pointer, which the client application uses to access the cached data. The practical *NetCache* limits for caching depend upon the amount of memory installed in the client machine. This memory could either be installed RAM or a partition of virtual memory on disk [KSL95]. The physical RAM will provide a much higher performance level than disk memory. In either case the performance will be better than the performance level of accessing the data

directly from the remote database. In its most simple form the pseudo-code of the algorithm that *NetCache* uses is as follows:

Algorithm Netcache
Input: A new Query Q_n.
The list of previously cached queries: ListQueries
Output: A pointer to a cached Q_n answer set.
Method
Begin
If (QueryMatch(ListQueries, Q_n)) **Then**
return (pointer)
Else
FetchFromDB(Q_n, pointer)
Return (pointer) /* produce a new pointer */
Endif
End Algorithm NetCache

The pertinent procedures which are used in the *NetCache* algorithm perform the following:

1. QueryMatch(ListQueries, Q_n): This procedure checks if query Q_n is the same as one of the already cached queries found in ListQueries.
2. FetchFromDB(Q_n, pointer): This procedure does the actual fetching of the answer sets to query Q_n and caches it in the memory. It assigns it a reference called a "pointer" for further access.

The architecture of *NetCache* is based on the Open Distributed Processing (ODP) model. At run-time the system decides whether to use the local service or the remote data service. The Open Distributed Processing (ODP) model includes the dynamic binding of clients to service end-points, which is a challenging issue for the new generation of telecommunication systems. The goal of the ODP model is to provide transparent sharing of services and resources over different architectures, networks and operating systems to hide the distribution of a system from its users.

In the ODP model, a client application calls a service Application Programming Interface (API) for a request. The application API invokes a trader. The trader[15] (who dynamically determines the best service to process the query) sets up the bindings and calls the server to process the request. This scenario is illustrated in Figure 6.1.

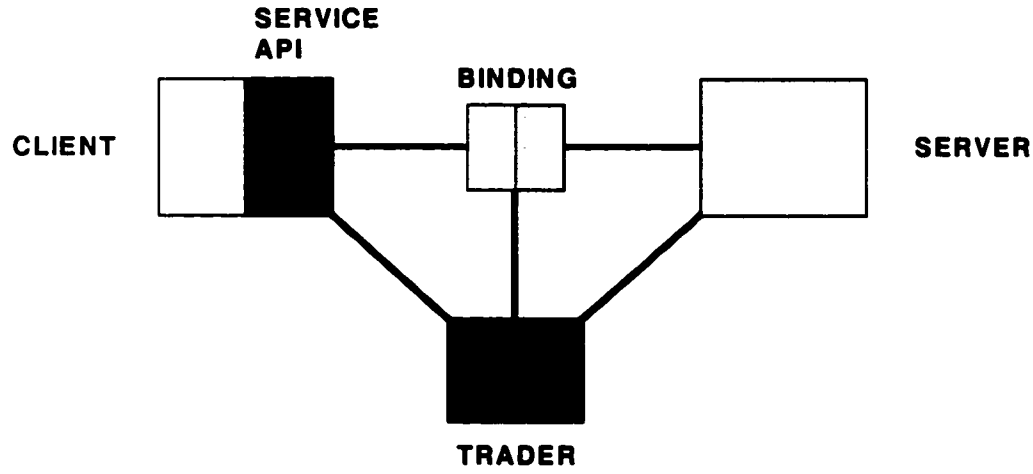


Figure 6.1: The ODP Model [KSL95]

In a *NetCache* ODP application, the client application invokes the *NetCache* service API to execute a query. The *NetCache* API calls the *NetCache* trader. If the answer to the query is found in local memory, the trader returns the table pointer to the *NetCache* API and the *NetCache* API returns the table pointer to the client. If the answer to the query is not already cached in local memory, the trader provides a binding to the remote database and the *NetCache* API caches the result in the local memory and returns the table pointer to the client.

[15] The trader is the object that enables the client to choose appropriate servers at run-time [KSL95]. The requestor of a service does not know the actual server identity in order to request the service. The trader is in charge of dynamically associating an appropriate binding among the objects involved in the service.

6.3 Related Works: Machine Learning Methods

We discussed, briefly, traditional query optimization and our new approach to this field in the context of distributed databases. We shall presently highlight the details of our solution, and the philosophy of how we make decisions on caching. Indeed, our solution fits into the general field of machine learning [DM83] and [Mic83], which is, concerned with the question of how to construct computer programs that automatically improve with experience.

Several fundamental types of inductive learning problems [DM83] have been studied for several years. They include classification, clustering and sequence prediction. In classification, the learning system is presented with (independent) instances representing a given class, and the task is to induce a general description of the class. The learned class description can be used to classify new instances whose correct class is not known and typical algorithms for this include decision tree algorithms (ID3 and C4.5) [Qui86], [Qui93] and Bayes classifier schemes [Pea88].

The second type of inductive learning problem is that of conceptual clustering [FD90]. Clustering problems arise when several objects are presented to a learner and the learner has to invent classes into which the objects can be usefully grouped.

The third type of inductive learning problem is that of sequence prediction. The sequence prediction problem is the one that primarily interests us when we are dealing with caching. This problem has been studied in the past as the problem of discovering a rule characterizing a given sequence of objects, and which can be used to predict the continuation of the sequence. Kotovsky and Simon [KS73] studied the problem of letter-sequence prediction where a user is given partial sequences of letters and is asked to predict the next few letters in the sequence. Their program finds a sequence-generating rule, which predicts the continuation of the sequence. It assumes that there is only one *correct continuation* of the sequence, and also that each object in the sequence has only one attribute, which is the name of that object. Related work on this type of learning is given in [Hof83]. Another related work is done by Dietterich and Michalski in [DM83] in which the authors present a method for discovering sequence prediction rules in cases where the objects in the sequence are described by many attributes and the sequence

prediction rule is non-deterministic. The learner is given a finite sequence of events E_1, E_2,..,E_N where each event is characterized by the values of a number of discrete-valued attributes a_1,a_2,..,a_M. The algorithm finds a *sequence generating rule* that given the first E_k events predicts the values of the attributes that must be true of event E_{k+1}. This sequence-generating rule may not predict a unique event k+1, and this makes the rule non-deterministic. The algorithm developed in this work is tailored specifically to the problem of rule discovery in the card game "Eleusis".

6.4 Caching using Polytrees

In this thesis we deal with a real-life application, and specifically in this application the only data or learning cases available is a huge trace of "Select" statements made by different users following a given application. This trace is considered as a sequence in which repeated patterns may exist. The aim is to capture the *repeated patterns* of queries so as to be able to perform anticipated caching. Our learning algorithm to solve this problem is done in two steps. The first step parses and generalizes the training examples of our application using inductive learning generalization methods. Using the results of the first phase, the second process builds a polytree, which represents most of the data dependencies. To achieve this the latter process uses the strategies proposed in the previous chapters. Each variable in this model is represented by a node, which is augmented with a set of conditional probabilities, thus forming the BBN.

To achieve this we first introduce the concept of a "Case". A **Case** represents a set of example queries made to the database(s) within a given time interval (or learning interval). During this time interval the observer can capture the requests made to distributed databases for a given application. Those cases will be used in our learning scheme as the training examples. In the specific real-life application we studied, this notion of a case is not really used but rather the collected queries form only one learning case. The example we are describing in the following section is to highlight the usage of distributed databases and to clarify the notion of columns and tables used in the select statements. Also we want to show that we can learn either from a set of learning cases or

from one single chain of consecutive Select statements. The latter represents the case of the real-life data used in this chapter.

6.5 Description of the Queries

Since we are dealing with transaction-based applications, which perform repeated access to the same data from remote relational database(s), we consider our database queries to be SQL-like (Structured Query Language) queries. These queries involve only *Select* statements, which fetch data from databases. SQL has one basic statement for retrieving information from a database namely, the *Select* statement. A *Select* statement has the following syntax:

Select <attribute list> **from** < table list> **where** <condition>

in which:

<attribute list>: is a list of attribute names whose values are to be retrieved by the query.

<table list> : is a list of the relation names required to process the query.

<condition> : is a conditional (Boolean) search expression that identifies the tuples to be retrieved by the query.

Whether the query retrieves all the tuples or only distinct tuples, the Select statement can also be written in the following form:

Select-Distinct(Attribute-List, Tables-List, **where**(condition)) or
Select-All(Attribute-List, Tables-List, **where**(condition)).

A trace of these SQL select statements constitutes our data for the real-life application considered in this chapter. This data is first parsed and then generalized into other select statements. The parsing and generalization processes are described in the following sections.

6.6 Parsing the queries

The parsing is considered the first step in the whole process of learning. Its purpose is to convert "raw" database data into a format that will be used by the generalization process in order to build the polytree structure. First of all, it "cleans" the data from the details of the compilation and execution of the queries and then replaces the implicit declarations of any parameter of the select statement with its corresponding attribute to ease the use of the data in the generalization process. For example, if a *Select* statement includes the '*" before the keyword "From", the parser will make the column parameter in that *Select* statement equal to all the columns of the database table. Note that the parsing process depends on the language we use to access the databases. In our specific application the language used is the PL/SQL which is the Oracle's procedural language extension to SQL. This language uses a construct called *cursor* to define a private SQL area and to access its stored information [SQL92]. The *cursor* can be implicit or explicit. Thus, when we execute a given query, we retrieve rows of data that meet the condition in the select statement. A query may return one or many rows of data. For queries that return more than one row of data, an explicit *cursor* can be declared in order to process the rows individually [SQL92]. The example below declares a *cursor* named c1 and how it is utilized.

Example 6.2:

We consider the following table called emptab which contains rows of data for the attributes employee number, last name, title and department with the corresponding variables empno, lname, title and dep for a certain organization respectively.

empo	lname	title	dept
1234	John	Professor	csi
2346	Russell	Professor	elg
5678	Smith	Student	csi
8965	Scott	Student	csi
7654	Robert	Student	csi
5643	Weston	Student	seg

Table 6.2: An Example of a Database Table

Suppose that in a given processing sequence we declare a *cursor* c1 to be the following query:

"Select empo, lname, title From emptab Where dept = csi"

The returned data from the execution of this query constitutes the *active set* of the data for the *cursor* c1. A cursor is first declared, opened and then closed. In Table 6.2, the active set for cursor c1 is the data given in Table 6.3 below:

1234	John	Professor
5678	Smith	Student
8965	Scott	Student
7654	Robert	Student

Table 6.3: Active Set for Cursor c1

A cursor is first positioned at the first row in the active set of rows. The subsequent fetching of the data will make the cursor progress to the next row in the active set as shown in Figure 6.2. Note that retrieving rows of data in the active set is completed by issuing the command "Fetch" with the *cursor* name as given in Figure 6.2.

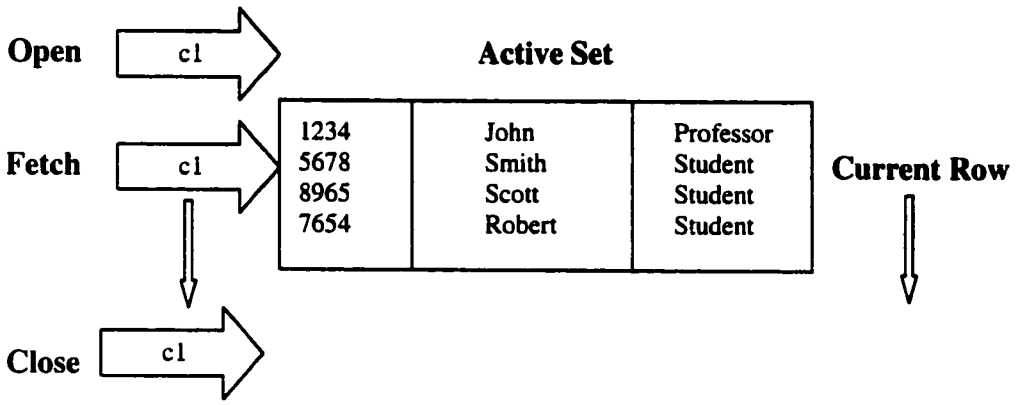


Figure 6.2: A Cursor Control

By declaring and opening many cursors, multiple queries can be processed in parallel. The trace of data we used in our application contains the commands related to the cursor management. Therefore the parsing process replaces all the fetching commands by the actual data. This process is very important since we are interested in collecting all the accesses made to remote databases. The main task of the parser is to store all the opened cursors with the corresponding queries, and then replace every call to that specific cursor by the corresponding query. An example of a file containing these cursors will be given in Section 6.9.2.

6.7 Generalization of the Queries

For our application, since the Select statements are simple expressions, we merely use the term "Generalization" for the process of obtaining a compact representation of the data and reduction in the size of the input trace. We also restrict our generalization to queries satisfying the properties given below. The Generalization process takes two *consecutive* queries and generalizes them into one single query. We say that two consecutive queries are "**generalizable**" if they have the same arguments i.e., the same columns and the same tables in their query form satisfying Definition 6.1 below. When we generalize the queries the column and table names will be considered equal for the respective queries, and will thus remain the same after the generalization. The generalization is achieved by

dropping the condition argument found in consecutive "where" clauses. The non-consecutive "where" clauses are dropped in the final stage when the dictionary of queries itself is built. The motivation for this Generalization process is that when accessing remote data, whether we retrieve one or more tuples from the databases, the most important feature is that we are processing that particular column of data found in that specific table.

Definition 6.1:

We say that two given queries

q_1= "Select (X1, from(Y1), where C1)" and

q_2= "Select (X2, from(Y2), where C2)"

are "**generalizable**" if and only if:

(X1 = X2 and Y1 = Y2).

In this case the generalized query is:

Gen= "Select (X1, from(Y1))" □

The generalization of queries within a case is done in an ordered basis. For example, consider a learning case from four queries a, b, c and d of the form:

Case = a b c d.

In this case the different generalizations which may occur are given below. The term Gen is used whenever a generalization is performed.

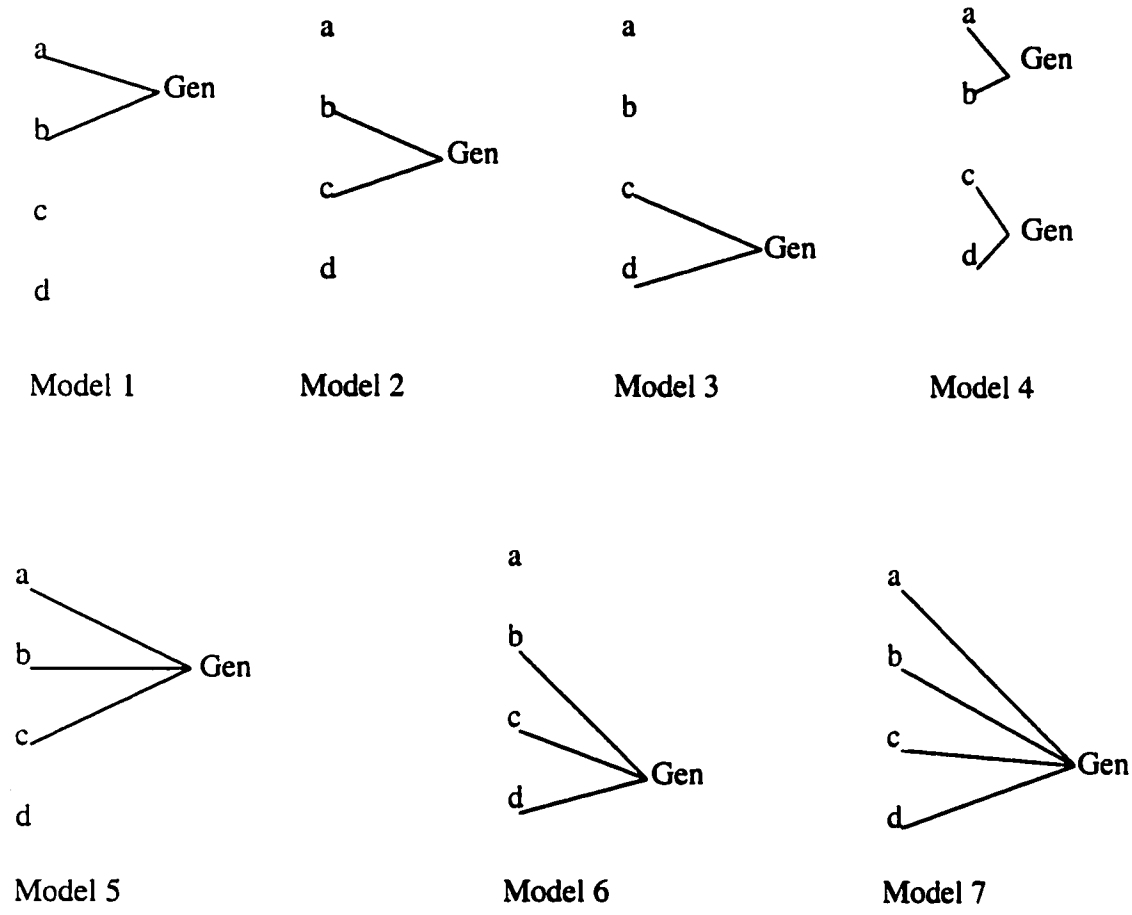


Figure 6.3: Possible generalizations within a case of four queries. Note that other combinations, e.g. (a, c) are not generalizable because they are not consecutive queries.

Figure 6.3 shows an example on how we can generalize the queries when we are given a case of four queries. Notice that in Model 4 queries a and b are generalizable as well as queries c and d. In this situation the set of the four queries is reduced to only two queries. In Model 7 all the four queries are generalizable and their generalization yields the single query Gen representing the case of queries a b c and d. In Model 1 only the first two

queries are possible for generalization. Similar situations are encountered for different variables in the other scenarios. This process of generalization is formally described below.

Algorithm Generalize-Queries
Input: A set of queries for a given learning case, say Case = q_1,..., q_n.
Output: Another set of queries gCase in which some queries may be generalized. The variable count is used as an index in the learning case.
Method:
Begin
Initialize count = 1
Initialize Gen = q_1
While (count ≤ n) **Do**
Get Next-Query = q_{count}
If (Gen and Next-Query are Generalizable) **Then**
Gen = **generalize**(Gen, Next-Query)
Else
Gen = Next-Query
EndIf
count = count + 1
Store Gen in gCase
EndWhile
End Algorithm Generalize-Queries

The goal of the generalization process is to reduce the input data by grouping consecutive fetches of the same columns of data from the same database tables into one single fetch of that data. The generalized queries are stored into a dictionary. They form the nodes of the polytree structure we build. To highlight the importance of this process we give the example below.

6.8 An illustrative Example

To explain the above procedures we consider a small example. We consider the databases that model an engineering company. The database tables shown in Tables 6.4 and 6.5 are taken from [OV91]. We imagine an application, which looks at the employee number and

subsequently gets the project and the associated budget with which that employee is involved.

The entities modeled are the engineers (ENG) and projects (J). For each engineer, we keep track of the employee number (ENO), name (ENAME), title in the company (TITLE), salary (SAL), identification number of the project the engineer is working on (JNO), responsibility within the project (RESP) and duration of the assignment (DUR) in months. Similarly, for each project we store the project number (JNO), the project name (JNAME), and the project budget (BUDGET). These relations are given in the tables below.

JNO	JNAME	BUDGET
j1	Instrumentation	150000
j2	Database Develop	135000
j3	CAD/CAM	250000
j4	Maintenance	310000

Table 6.4: The Budget Table [OV91]

ENO	ENAME	TITLE	SAL	JNO	RESP	DUR
E1	J.DOE	Elect.Eng	40000	j1	Manager	12
E2	M.Smith	Analyst	34000	j1	Analyst	24
E2	M.Smith	Analyst	34000	j2	Analyst	6
E3	A.Lee	Mech.Eg	27000	j3	Consultant	10
E3	A.Lee	Mech.Eng	27000	j4	Engineer	48
E4	J.Miller	Programmer	24000	j2	Programmer	18
E5	B.Casey	Syst.Anal	34000	j2	Manager	24
E6	L.Chu	Elect.Eng	40000	j4	Manager	48
E7	J.Jones	Syst.Anal	34000	j3	Manager	40

Table 6.5: The Engineer Table [OV91]

In order to build some learning cases for the application, different users have been asked to issue queries to get the project they are working on and the corresponding budget given their employee numbers. Every set of queries made to the database tables form a learning case for the application. The result of these learning cases are given in Table 6.5 below:

case	Query 1	Query 2	Query 3	Query 4
1	Select DISTINCT eno from Eng	Select * from Eng where eno = E3	Select * from Eng where ((eno=E3) AND (resp = Consultant))	Select jname, budget from Budgets where jno = j3
2	Select DISTINCT eno from Eng	Select * from Eng where eno = E7	Select jname, budget from Budgets where jno = j3	
3	Select * from Eng where eno = E4	Select jname, budget from Budgets where jno = j2		
4	Select * from Eng where eno = E2	Select jname, budget from Budgets where ((jno=j2) or (jno=j1))		
5	Select DISTINCT eno from Eng	Select * from Eng where eno = E5	Select jname, budget from Budgets where jno = j2	
6	Select DISTINCT eno from Eng	Select * from Eng where eno = E6	Select jname, budget from Budgets where jno = j4	
7	Select DISTINCT eno from Eng	Select * from Eng where eno = E8	Select jname, budget from Budgets where jno = j3	

Table 6.6: Query Cases encountered for the Budget and Engineer Tables

We first consider the queries made in case 1:

Case 1:

Query 1: "Select Distinct eno from Eng;"
Query 2: "Select * from Eng where eno = E3;"
Query 3: "Select * from Eng where ((eno=E3) AND (resp = Consultant));"
Query 4: "Select jname, budget from Budgets where jno = j3;"

After the parsing process, the set of queries in Case 1 can be compacted as below. Observe that in the interest of simplicity we have omitted the term "From" from the *Select* statements and instead put the lists of columns and tables in square brackets.

Case 1:

Query 1: Select-Distinct([eno], [Eng])
Query 2: Select-All(all columns,[Eng],where(eno = E3))
Query 3: Select-All(all-columns,[Eng], where(((eno=E3) and (resp = Consultant)))
Query 4: Select-All([jname, budget], [Budgets], where(jno = j3))

Following the generalization algorithm, while Query 1 and Query 2 in Case 1 cannot be generalized in the sense of Definition 6.1, Query 2 and Query 3 will be generalized into the following Query:

Gen = Select-All(all columns, [Eng])

Again query Gen and Query 4 cannot be generalized. So the result of the generalization algorithm within Case 1 is the following:

Case 1:

Query 1 : Select-Distinct([eno], [Eng])
Query 2 : Select-All(all-columns, [Eng])
Query 3 : Select-All([jname, budget], [Budgets], where(jno = j3)).

The same process of generalization is done on the other cases and yields the following queries:

case	Query 1	Query 2	Query 3
1	Select-Distinct([eno], [Eng])	Select-All(all-columns, [Eng])	Select-All([jname, budget], [Budgets], where(jno = j3))
2	Select-Distinct([eno], [Eng])	Select-All(all-columns, [Eng],where(eno = E7))	Select-All([jname, budget], [Budgets], where(jno = j3))
3	Select-All(all-columns, [Eng], where(eno = E4))	Select-All([jname, budget], [Budgets],where(jno=j2))	
4	Select-All(all-columns, [Eng], Where(eno=E2))	Select-All([jname, budget], [Budgets], where((jno=j2) or (jno=j1)))	
5	Select-Distinct([eno], [Eng])	Select-All(all-columns, [Eng], Where(eno=E5))	Select-All([jname, budget], [Budgets], Where(jno=j2))
6	Select-Distinct([eno], [Eng])	Select-All(all-columns, [Eng], Where(eno=E6))	Select-All([jname, budget], [Budgets], Where(jno=j4))
7	Select-Distinct([eno], [Eng])	Select-All(all-columns, [Eng], where(eno=E7))	Select-All([jname, budget], [Budgets], where(jno=j3))

Table 6.7: Generalized Query Cases

We can now define a dictionary of the generalized queries as:

q1: Select-Distinct([eno], [Eng])

q2 : Select-All(all-columns, [Eng])

q3 : Select-All([jname, budget]), [Budgets])

Using the set of the generalized queries defined in the dictionary, the previous learning cases can be written as the following:

The learning Cases after generalization are the following:

Case 1: q1 q2 q3
Case 2: q1 q2 q3
Case 3: q2 q3
Case 4: q2 q3
Case 5: q1 q2 q3
Case 6: q1 q2 q3
Case 7: q1 q2 q3

These sets of queries will now form the learning cases that we use to build our polytree structure. Notice that in our real-life application, the learning cases are not available but rather a single trace of data is provided to us from which we intend to learn the polytree structure. We tried first to divide this trace of data into several learning cases but as the decision as to where the splits should be done was not clear, we opted to process the whole trace as a single learning case and build the polytree structure from the computed information measure between the generalized queries present in that trace of data. We computed the conditional probabilities from the trace and deduced the information measure between every pair of variables.

6.9 Real-Life Application: Distributed Database

6.9.1 Description of the Application

The application we are studying includes clients and a distributed database server. These clients make repeated accesses to data stored at this server. They follow an underlying application, which was initially unknown to us at the time we processed the data and implemented our algorithms. The problem was to learn the usage pattern of this distributed database.

The stream of SQL queries is taken from a database server where the client application is used by clerks to manage the assignment of physical telephones to organizations within the National Capital Regions for the Department of National Defence of the Government of Canada. The database tracks the telephones numbers assigned to the various telephones, the physical location of the handsets and connections, and the organization to which the telephones have been assigned.

There are 32 clerks. After examining the trace data, we observed that these clerks put approximately 121181 queries against the database in the duration of the trace, which was 24 hours. They make concurrent access to this database 24 hours per day and 7 days per week.

6.9.2 Parsing of the Queries

- **Data Collection**

The data or the input file to the project is a file containing a real trace of the *Select* statements (queries) made to the distributed databases for a period of 24 hours. The size of the trace file is 36MB. This file also contains all the details related to the execution of the *Select* statements. A small sub-file of this file is given as Example 6.3 and some typical statements contained in it are given below.

- **Parsing the Data**

The data as it is given in the trace file is a set of execution of the *Select* statements. This execution used the notion of a Cursor to declare a *Select* statement, and to invoke it from any location at a subsequent stage. The data used in the *Select* statement represented by the given cursor is fetched at any time by invoking the command "Fetch" with the cursor number. The parsing process analyzes the cursor statements and stores every declaration of a cursor to access the data used by the *Select* statement defined in that cursor. The examples below give a portion of data from the initial trace of the accesses to the database. In this example two cursors were declared which are cursor #1 and cursor #3.

Example 6.3: Parsing

```
=====================
PARSING IN CURSOR #1 len=147 dep=1 uid=0 oct=3 lid=0 tim=4092015242
select privilege#,level from sysauth$ connect by grantee#=prior privilege# and privilege#>0 start
with (grantee#=:1 or grantee#=1) and privilege#>0
END OF STMT
PARSE #1:c=1,e=1,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=4092015242
EXEC #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=4092015242
FETCH #1:c=0,e=0,p=0,cr=4,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015242
FETCH #1:c=0,e=0,p=0,cr=5,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015242
FETCH #1:c=0,e=0,p=0,cr=7,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015242
FETCH #1:c=0,e=0,p=0,cr=5,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015242
FETCH #1:c=0,e=1,p=0,cr=5,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015243
FETCH #1:c=0,e=0,p=0,cr=5,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015243
FETCH #1:c=0,e=0,p=0,cr=5,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015243
FETCH #1:c=1,e=0,p=0,cr=3,cu=0,mis=0,r=0,dep=1,og=4,tim=4092015243
=====================
PARSING IN CURSOR #3 len=36 dep=1 uid=0 oct=3 lid=0 tim=4092015244
select text from view$ where obj#=:1
END OF STMT
PARSE #3:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=4092015244
EXEC #3:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=4092015245
FETCH #3:c=1,e=0,p=1,cr=4,cu=0,mis=0,r=1,dep=1,og=4,tim=4092015245
=====================
```

The result of parsing this portion of input data is given below. The cursor is declared and its corresponding query is stored in memory.

Example 6.3: Fetching

```
=====================
cursor: #1
select: select level, privilege# from sysauth$ connect by grantee# = prior privilege# and privilege#
> 0 start with ( grantee# = : 1 or grantee#=1 ) and privilege# > 0
select key string: level, privilege# from sysauth$
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
fetch(#1) == 1
=====================
cursor: #3
select: select text from view$ where obj# = : 1
select key string: text from view$
fetch(#3) == 8
=====================
```

The numbers introduced and assigned here to the fetching command "fetch(#)" represent the addresses of the locations where the actual queries are saved and stored.

6.9.3 Generalization of the Queries

In order to generalize the queries obtained after the parsing process we used the algorithm Generalize-Queries for queries satisfying Definition 6.1. This process reduces the number of queries accessing the same columns of data in the same tables. We reduced the number of queries in the trace from 121,181 queries to a sequence of 31,978 queries. A portion of data taken from the parsed file of the trace, which includes candidates for generalization, is given below. This example gives some generalizable statements. After the generalization process every query is assigned two given addresses. The first one is the address already assigned in the parsing process, which is the address of the initial query. The second address is the address of the location for the generalized query in the dictionary of the generalized queries. In the example below, the number 280 represent the index of the location where the generalized statement is stored and the number 3156 is the address where the initial query containing the "where" clause in its *Select* statement is stored.

The following queries are generalizable queries.

Example 6.3: Generalizing

280 3156

Select account_bill_num, account_holder_uic, comm_group_id, cost_centre_code, facility_number, facility_type, gl_acct_code, item_completion_date, item_seq_num, quantity, rate_group, recoverable_code, recoverable_mrc, rowid, tso_equipment_code, user_stn_seq_num, vendor_equip_code, vendor_ident **From** equipment_item **Where** (comm_group_id = '76') and (account_holder_uic = '0001') and (account_bill_num = ' 1061') and (facility_type = 'LOCAL') and (facility_number = '9957848') and (user_stn_seq_num = '001') order by item_seq_num

280 3152

Select account_bill_num, account_holder_uic, comm_group_id, cost_centre_code, facility_number, facility_type, gl_acct_code, item_completion_date, item_seq_num, quantity, rate_group, recoverable_code, recoverable_mrc, rowid, tso_equipment_code, user_stn_seq_num, vendor_equip_code, vendor_ident **From** equipment_item **Where** (comm_group_id = '76') and (account_holder_uic = '0001') and (account_bill_num = ' 1061') and (facility_type = 'LOCAL') and (facility_number = '9959156') and (user_stn_seq_num = '001') order by item_seq_num

280 3148
Select account_bill_num, account_holder_uic, comm_group_id, cost_centre_code, facility_number, facility_type, gl_acct_code, item_completion_date, item_seq_num, quantity, rate_group, recoverable_code, recoverable_mrc, rowid, tso_equipment_code, user_stn_seq_num, vendor_equip_code, vendor_ident **From** equipment_item **Where** (comm_group_id = '76') and (account_holder_uic = '0001') and (account_bill_num = ' 1061') and (facility_type = 'LOCAL') and (facility_number = '9963770') and (user_stn_seq_num = '001') order by item_seq_num

After the generalization process some statistics are derived. The following example is a portion of data taken from the file containing these results. The first number in every line represents the number of the consecutive generalized statements, the fetch command includes the number of the cursor it is fetching from and the index of the generalized query which corresponds to that cursor.

Example 6.3: Getting the learning trace

(1) fetch(cursor #69) = 2389
(1) fetch(cursor #70) = 2390
(2) fetch(cursor #71) = 1243
(2) fetch(cursor #72) = 1669
(2) fetch(cursor #73) = 510
(2) fetch(cursor #34) = 1668
(1) fetch(cursor #74) = 1671
(2) fetch(cursor #35) = 91
(2) fetch(cursor #35) = 91
(2) fetch(cursor #35) = 91
(2) fetch(cursor #72) = 1669
(2) fetch(cursor #73) = 510
(2) fetch(cursor #34) = 1668
(2) fetch(cursor #35) = 91
(2) fetch(cursor #35) = 91
(2) fetch(cursor #35) = 91

The obtained generalized queries, which are stored in a dictionary, are considered as nodes of the structure which we intend to build from the dictionary itself.

6.9.4 Building the Chow Tree

After the parsing and the generalization processes, the initial trace of the select statements is reduced to a sequence of numbers. The number of the select statements in the processed dataset was 31,978, which is mainly the sequence of the repeated generalized select statements in the trace. This sequence contains numbers that represent indexes in

the dictionary where the generalized *Select* statements are stored. It contained repeated patterns, which was to be detected by our algorithms. The total number of these generalized queries in the dictionary is 624, which represents also the number of the nodes in the structure we build. Using the sequence of the queries, we computed the conditional probabilities of the nodes and then from these probabilities we derived the information measures between every pair of nodes. The generalized *Select* statements are the actual nodes of the Chow tree.

Example 6.8: The queries representing the nodes of the tree given in Figure 6.5.

3249 select :b1, :b2 from dual
1245 select facility_type from lf_type_util
1244 select account_bill_num from leased_facility
114 select nvl(0, sum(mrc_encumbered)), nvl(0, sum(nrc_encumbered)) from order_equipment_item
421 select ', cfcc_fin_resp_uic e' fr from dual
3408 select '1' from expenditure
273 select 'x' from financial_code
217 select 'x' from leased_facility
332 select 'x' from uic_util
271 select 'x' from monthly_recur_charge
119 select account_holder_uic, comm_group_id, contact_name, from_work_site_address, from_work_site_bldg, from_work_site_floor, from_work_site_room, inv_user_stn_seq_num, lf_seq_num, order_user_seq_num, originator_ext, originator_fy, originator_serial, phone, phone_ext, rowid, to_work_site_address, to_work_site_bldg, to_work_site_floor, to_work_site_room, uic_id, user_name, user_seq_num, user_uic from order_user_station

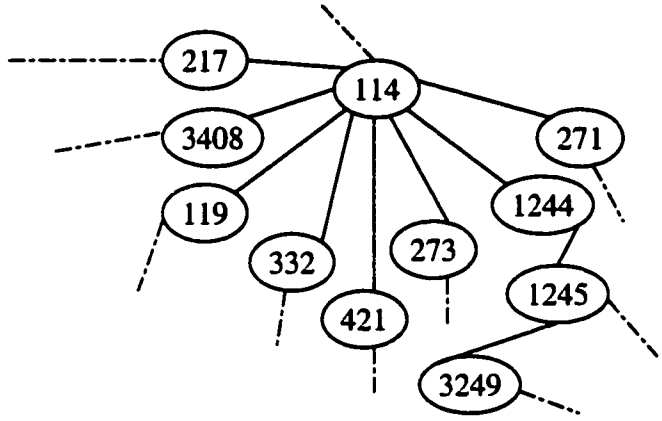


Figure 6.4: A Portion of the Chow tree obtained from the trace of data

6.9.5 Orienting the Polytree

The orientation of the polytree is based on independence of nodes, which are multiple parents of a given node in the tree structure. In order to get independence we used the correlation coefficient as explained in Section 5.3. These correlation numbers are clearly not equal to zero, but we used a threshold for independence. The independence decision depends on the threshold used to orient the polytree. The following figure (Fig. 6.5) shows a portion of the polytree obtained using a value of 0.001 as a threshold for getting independence.

All the input files provided to this application, the parsed files, the generalized files and the complete polytree are found in the following URL: www.site.uottawa.ca/~ouerd

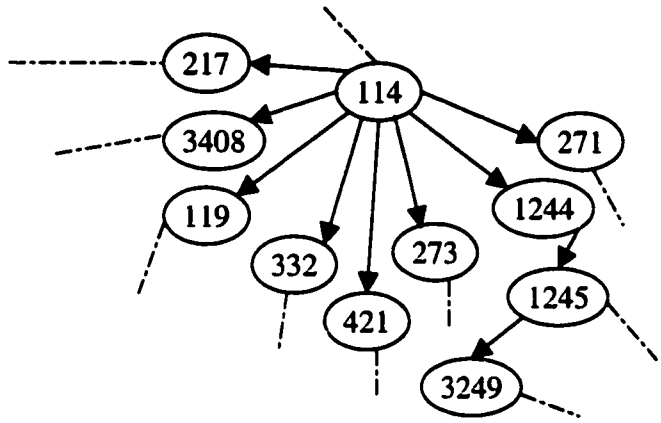


Figure 6.5: A Portion of the Polytree obtained from the trace of data

6.10 Verification of the Polytree[16]

As in any real-life application, the verification of the results is the most difficult part of the entire exercise. In this case, the verification of the quality of the polytree obtained is a problem in itself. Indeed, in the words of King [Kin99], the expert from whom the data was obtained, this could be a thesis in its own right [Kin99]. Quite simply put, King approved the results very positively. He adds: "you have shown that requests are reused and that the response-time will be improved by using your algorithms. Cleanly and optimally implementing these algorithms and integrating them with NetCache is a research and development project; with an emphasis on research and the uncertainty of exactly how successful the effects will be on improving the performance of *NetCache*-enabled applications. If very successful, as your research suggests, then the product might be marketable".

One possible approach for such a testing strategy would be cross-validation, but even here, the method by which this can be achieved is open.

To estimate the effect of our strategy on performance, one would require implementation of the system in conjunction with *NetCache*. Such a study is far beyond

[16] Many thanks to Dr. Douglas King for providing the data for this application and also verifying the results.

the scope of the present thesis. Apart from the problem being conceptually difficult, King has enumerated numerous anticipated difficulties of such a project. These difficulties include modeling the following:

- A real application streams of SQL requests and the true amount of returned data. Such a model would include the timing dependence of the SQL transmission, preparation, and execution of the database engine,
- The amount of memory consumed by the software implementing the algorithms, which again would include the timing dependence of this usage,
- The network environment and overheads in production environments (e.g., Internet), and
- The amount of CPU and other system resources consumed by the software implementing the algorithms, and the time-driven aspects of this resource consumption.

The tasks of building causal structures and the process of integrating them with *NetCache* is considered as a future work.

6.11 Summary

In this chapter we have described the problem of query optimization in distributed databases. We showed that learning the workflow of the data could reduce the communication time. We considered a real-life application on which we applied the algorithms discussed in the previous chapters. We also gave an overall sketch of our solution to the query optimization problem when we are dealing with distributed databases. Our approach to the problem is to reduce the volume of queries made to distant databases especially for applications that make repeated accesses to the same relations stored in remote databases. By introducing the notion of caching we try to take advantage of performing local accesses rather than remote accesses, because the former significantly reduces the communication time, and thus improves the overall performance of the system.

We also described the various algorithms needed to process the queries in order to build the training cases for the learning procedure to be used for the caching. The initial

training set is the trace of the queries made against the distributed databases. This set of queries cannot be used directly by our learning algorithms. It is first parsed and written in a form that is appropriate for the next process, which is the generalization process. The generalization process reduces the number of queries in the trace. The resultant queries form the nodes of the goal structure. Using the information measure between every two pairs of nodes we used our algorithms introduced in Chapters 3-5 to deduce dependencies among these variables and thus determine the polytree structure. The problem of running the Bayesian method introduced in Chapter 2 given by [CH92] on our real-life data remains an open problem. In fact, the K2 algorithm learns from a set of learning samples and in our application the decision on how we could divide the whole trace of data into learning cases is itself an open problem.

A summary of the results of this chapter is currently being compiled as a publication [OMO99].

Chapter 7

Conclusion

The problems studied in this thesis are related to knowledge representation. In this thesis we have considered methods for building Bayesian belief networks from samples. We studied the problem of approximating distributions by a polytree-based distribution. We applied the theoretical results we have obtained to a real-life application of increasing performance in systems using distributed databases. We primarily considered applications that make repeated accesses to the same data stored in remote databases. Our approach to tackle this problem was to learn from the accesses to these databases so as to be able to deduce causal relationships among queries made to the databases and build rules to allow decisions on caching repeated patterns of queries.

This chapter summarizes the results presented in this thesis and suggests future work. We shall cover the problems in the same order as they have been presented in this thesis.

In Chapter 2, we studied the problem of representing knowledge. We presented some related work to the problem of constructing a network automatically from direct empirical observations. Bayesian approaches to learn the graphical structure of Bayesian Belief Networks (BBN) from databases and probabilistic methods for building BBNs are presented. In building structures, we have also studied the problem of approximating densities for discrete and continuous random vectors.

In Chapter 3, we considered the problem in which we know the probability distributions and we are trying to derive an adequate representation for these distributions. We studied a specific algorithm developed by [Pea88] for approximating distributions by a polytree structure. The algorithm assumes that the underlying distribution is a polytree based distribution and uses the Chow tree method to build the tree; the skeleton of the polytree. To orient the tree, this algorithm uses independence

assertions, which assert that two parents of a given node are marginally independent. Our contribution here was to consider a Depth First Search (DFS) traversal to orient the tree for both discrete and continuous cases, and to study the complexity of these algorithms. These algorithms have been implemented.

In Chapter 4, we developed a sampling algorithm, which generates data from a given polytree. We studied the problem of how can we obtain a valid *distribution* that the polytree structure will allow. We have developed these algorithms for both discrete and continuous cases. For the discrete case we considered and implemented two methods: a method when the first order probabilities were given and a method when the conditional probabilities on the nodes are given. For the continuous case, the algorithm generates the samples from an underlying multi-variate Normal distribution whose parameters μ and Σ are known.

In Chapter 5, we considered in details how to derive the independence tests that we were using in the polytree algorithm to orient the tree. Since we are dealing with a potentially real-life application, our data does not necessarily support statistical independence even though the random variables are statistically independent. We therefore used the correlation coefficient to infer whether two random variables are independent instead of the information metric I_f. Uncorrelation can be seen to be an approximation to independence when we are dealing with discrete random variables. If however we consider continuous normal variables, independence of two variables is exactly equivalent to their being uncorrelated. In the case of discrete variables we used the correlation coefficient to infer independence of two variables. The main advantage of the correlation coefficient is that it is computationally less intensive than the information metric.

We also propose to use other tests to decide the independence of variables. These tests include the Chi-square hypothesis test when we know in advance the number of samples.

In this chapter we have also developed algorithms for building polytrees when the joint or marginal probability distributions are not explicitly given but only samples $\mathcal{S}$ are available from the application domain. In this case we estimate the dependence polytree

from the set of samples. In fact, in practical applications probability distributions are not available but only frequencies are obtained using the sample data. We considered a real-life application (Alarm Network) and run most of our algorithms on this data.

In Chapter 6, we described the problem of query accessing when we deal with distributed databases. We presented the query optimization approach and our solution strategy to deal with repeated queries to the same data in remote databases.

We developed the algorithms that process the queries. First we parsed the queries which are written in SQL-like language. These queries are basically the select statements, which retrieve column data from remote tables. We introduced the notion of generalization of queries. This generalization process compresses the information found in the queries. We finally built the Chow tree and the polytree from the dictionary of the generalized queries.

Future Work

As a future work we intend to complete the following evaluation (a kind of 4–fold cross validation) on our system:

- Train the system on ¾ of the sequences and,
- Test the system on the remaining ¼ of the sequences.

For each sequence of queries $q_1,..,q_N$ we will see if the learner network predicts each q_i given $q_1,..,q_{i-1}$. The question of how it predicts correctly the next query remains an open problem.

A practical use of the concepts introduced in this thesis would involve the Internet. Most of the Internet applications would benefit from similar caching arrangements so as to improve their performance. Repeated pages can be cached in a proxy server and thus avoid using the remote server. The problem here is one of minimizing the number of pages which fetch data from distributed servers by caching the repeated pages, and thus avoiding the re-fetching of the same data and allowing the system to anticipate the fetching of new data.

Another possible avenue for future work is the generalization process described in Chapter 6. In the generalization process, as it is currently described in Chapter 6, we

generalized two or more consecutive queries when they access the same columns of data from the same tables. Two given queries q_1= Select (X1, from(Y1), where C1) and q_2= Select (X2, from(Y2), where C2) are "generalizable" if and only if: (X1 = X2 and Y1 = Y2). As a future work we believe it is possible to weaken the condition in the definition of generalizable queries to be able to compress a larger set of data. We propose to consider two queries generalizable if (X1 ≡ X2 and Y1 ≡ Y2) where the symbol ≡ could imply, for example, inclusion. This new definition will probably yield a superior compression of the training data and build smaller Bayesian network.

Apart from the above we also propose that the concepts introduced here can be used to handle a broader class of applications. As an open problem we would like to see if the algorithms developed in this work can be used for software testing, where the problem involves predicting the set of important tests, from an ensemble of tests, that a user should perform to any code given what it has already performed in the past. Also our system is appropriate whenever data access performance is important. One example is the use of data in list-boxes in a user interface. User response-time is critical in the user interface, and the data is often semi-static. The query associated with the list-box can be pre-fetched and all subsequent displays of the list-box will use data that has been stored in local memory. Another example is the use of lookup tables to convert item codes to data values. Again, the data can be pre-fetched so that later accesses are from local memory instead of from the remote data server.

Another practical use of our system is in the operating systems design where accesses to memory when dealing with page faults are required. Instead of using the LRU or FIFO we can actually try to learn from the history and guess the best strategy for the caching of the loading of the pages.

References

[BKRK97] Binder, J., Koller, D., Russell, S. and Kanazawa, K., (1997). Adaptive Probabilistic Networks with Hidden Variables. Machine Learning, 29, 213-244.

[BSCC89] Beinlich, I.A., Suermondt, H.J., Chavez, R.M. and Cooper, G.F., (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In: Proceedings of the Second European Conference on Artificial Intelligence in Medicine, London, England, 247-256.

[Bun94] Buntine, W.L., (1994). Operations for Learning with Graphical Models. Journal of Artificial Intelligence Research, 2:159-225.

[Cha91] Charniak, E., (1991). Bayesian Networks without Tears. AI Magazine 12: 50-63.

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

[CH94] Chickering, D.M. and Heckerman, D., (1994). Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, Microsoft Corporation.

[CWS79] Chow, C.K., Wang, S.S.M., and Siegel, J.H., (1979). Sequential Classification of Patient Recovery Patterns After Coronary Artery Bypass Graft Surgery. Computers and Biomedical Research,12: 589-613.

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

[Coo87] Cooper, G. F., (1987). Probabilistic Inference Using Belief networks is NP-Hard. Technical Report, KSL-87-27, Medical Computer Science Group, Stanford Univ.

[Dat86] Date, C. J., (1986). An Introduction to Database Systems. Volume 1, Fourth Edition, Addison-Wesley Publishing Company.

[Das99] Dasputa, S., (1999). Learning Polytrees. Uncertainty in Artificial Intelligence.

[Die97] Dietterich, T. G., (1997). Machine-Learning Research, AI Magazine, Winter 1997.

[DM83] Dietterich, T.G, and Michalski, R.S., (1983). Learning to Predict Sequences. Machine Learning II: An Artificial Intelligence Approach. Edited by Michalski, Carbonell and Mitchell.

[EN94] Elmasri, R., and Navathe, S.B., (1994). Fundamentals of Database Systems. Second Edition. The Benjamin/Cummings Publishing Company.

[FD90] Fisher, D.H., (1990). Knowledge Acquisition via Incremental Conceptual Clustering. In Reading in Machine Learning. Edited by Jude W.Shavlik and Thomas G. Dietterich.

[FG96] Friedman, N., and Goldszmidt, M., (1996). Building Classifiers Using Bayesian Networks. In Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence. Menlo Park, Calif.: 1277-1284.

[FNP99] Friedman, N., Nachman, I., and Pe'er, D., (1999). Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. Uncertainty in Artificial Intelligence.

[Fri98] Friedman, N., (1998). The Bayesian Structural EM Algorithm. Uncertainty in Artificial Intelligence.

[Gov97] T36E Class Notes, (1997). Design and Implementation of Internet Applications. The Institute for Government Informatics Professionals.

[GPP90] Geiger, D., Paz, A., and Pearl, J., (1990). Learning Causal Trees from Dependence Information. Proceedings of AAAI, Boston, MA: MIT Press, 770-776.

[GVP90] Geiger, D., Verma, T. and Pearl, J., (1990). d-separation : from Theorems to Algorithms. In M.Henrion, R.D. Shachter, L.N. Kanal, & J.F. Lemmer (Eds), Uncertainty in Artificial Intelligence 5. Amsterdam: North-Holland.

[Heb92] Hebrawi, B., (1992). OSI Upper Layer Standards and Practices. McGraw-Hill Series on Computer Communications.

[Her91] Herskovits, E.H., (1991). Computer-Based Probabilistic-Network Construction, Doctoral dissertation, Medical Information Sciences, Stanford University.

[HGC95] Heckerman, D., Geiger, D., and Chickering D.M., (1995). Learning Bayesian Networks: the combination of Knowledge and Statistical Data. In Machine Learning, 20: 197-243.

[Hen88] Henrion, M., (1988). Propagating Uncertainty in Bayesian Networks by Logic Sampling. In J.F. Lemmer & L.N. Kanal (Eds). Uncertainty in Artificial Intelligence 2. Amsterdam: North-Holland.

[Hof83] Hofstadter, D.R., (1983). The Architecture of JUMBO. Proceedings of the International Machine Learning Workshop, R.S. Michalski (Ed), Allerton House, University of Illinois at Urbana-Champaign, June 22-24, 161-170.

[IK84] Ibaraki, T., and Kameda, T., (1984). On the Optimal Nesting Order for Computing N-Relation Joins. ACM Transactions database Systems 3: 482-502.

[Kin99] King, D., (1999). Personal Communication. President and CEO, KSL King Systems Limited, 28 Newton Street, Ottawa, Ontario, Canada, K1S 2S7.

[KL51] Kullback, S., and Leibler, R.A., (1951). On Information and Sufficiency. Annual Mathematics Statistics, 22: 79 – 86.

[KSL95] KSL King Systems Limited. (1995). The NetCache Software Manual, The Remote Database Performance, NC01.

[KS73] Kotovsky, K., and Simon, H. A., (1973). Empirical Tests of a Theory of Human Acquisition of Concepts for Sequential Patterns. Cognitive Psychology, 4: 399-424.

[Mit97] Mitchell, Tom , M., (1997). Machine Learning. McGraw-Hill

[Mei99] Meila, M., (1999). An Accelerated Chow and Liu Algorithm: Fitting Tree Distributions to High-Dimensional Sparse Data. ECML 1999, 249 - 257.

[Mic83] Michalski, R.S., (1983). A Theory and Methodology of Inductive Learning. Artificial Intelligence, 20:111-161.

[OV91] Ozsu, M.T., and Valduriez, P., (1991). Principles of Distributed Databases Systems. Prentice Hall.

[OOM99A] Ouerd, M., Oommen, J., and Matwin, S., (1999). Inferring Statistical Dependence for Random Vectors given Skeletal and Pairwise Independence Information. In preparation.

[OOM99B] Ouerd, M., Oommen, J., and Matwin, S., (1999). Generation of Random Vectors for Underlying DAGs Structures given First Order Marginals. In preparation.

[OOM99C] Ouerd, M., Oommen, J., and Matwin, S., (1999). Inferring Polytree Dependencies from Sampled Data. In preparation.

[OMO99] Ouerd, M., Matwin, S., and Oommen J., (1999). Applying Polytree Representations for Telephony. In preparation.

[Pea88] Pearl, J., (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA.

[PV87] Pearl, J., and Verma, S.T., (1987). The Logic of Representing Dependencies by Directed Acyclic Graphs. AAAI: 347-379, Seattle Washington.

[Qui86] Quinlan, J.R., (1986). Induction of Decision Trees. Machine Learning, 1:81-106.

[Qui93] Quinlan, J.R., (1993). C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufman.

[RS97] Ramoni, M., and Sebastiani.,P. (1997). Learning Bayesian Networks from Incomplete Databases. Technical Report Number, KMI-TR-43, Knowledge Media Institute. The Open University. Walton Hall, Milton Keynes, United Kingdom.

[RP87] Rebane, G., and Pearl, J. (1987). The Recovery of Causal Polytrees from Statistical Data. Proceedings of the Workshop on Uncertainty in Artificial Intelligence (pp. 222- 228). Seattle, Washington.

[Rob77] Robinson, R.W., (1977). Counting Unlabeled Acyclic Digraphs. In C.H.C. Little (Ed.), Lecture notes in mathematics, 622: Combinatorial Mathematics V. New York: Springer-Verlag.

[SQL92] PL/SQL User's Guide and Reference, Version 2.0, (1992). Oracle Corporation.

[Sky80] Skyrms, B., (1980). Causal Necessity, Yale University Press, New Haven.

[SGS90] Spirtes, P., Glymour, C., and Scheines, R., (1990). Causality from Probability. In G. McKee (Ed.), Evolving Knowledge in Normal and Artificial Intelligence. London: Pitman.

[SRA90] Srinivas, S., Russell, S. and Agogino, A., (1990). Automated Construction of Sparse Bayesian Networks from Unstructured Probabilistic Models and Domain Information. In M.Henrion, R.D. Shachter, L.N. Kanal, & J.F. Lemmer (Eds), Uncertainty in Artificial Intelligence 5. Amsterdam: North-Holland.

[Sup70] Suppes, P., (1970). A Probabilistic Theory of Causality, North-Holland, Amsterdam.

[Val91] Valiveti, R.S., (1991). Ph.D Thesis. Learning Algorithms for Data Retrieval and Storage. Carleton University. Ottawa. Canada.

[VO92] Valiveti, R.S. and Oommen, B.J., (1992). On Using the Chi-Squared Metric for Determining Stochastic Dependence. Pattern Recognition 11:1389-1400.

[VO93] Valiveti, R.S., and Oommen, B.J., (1993). Determining Stochastic Dependence for Normally Distributed Vectors using the Chi-Squared Metric. Pattern Recognition 6: 975-987.

[VP91] Verma, TS., and Pearl, J., (1991). Equivalence and Synthesis of Causal Models. In P.P. Bonissone, M.Henrion, L.N. Kanal, & J.F. Lemmer (Eds), Uncertainty in Artificial Intelligence 6. Amsterdam: North-Holland.

[VP88] Verma, T.S., and Pearl, J. (1988). Causal Networks: Semantics and Expressiveness. In Fourth Workshop on Uncertainty in AI St. Paul Minnesota: 352-359.