Belief Network Induction

by

Charles Ronald Musick Jr.

B.S. (University of Illinois at Urbana-Champaign) 1988

A dissertation submitted in partial satisfaction of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Stuart J. Russell, Chair
Professor John Rice
Professor Umesh V. Vazirani

1994

The dissertation of Charles Ronald Musick Jr. is approved:

Chair Date

Date

Date

University of California at Berkeley

1994

Belief Network Induction

by

Charles Ronald Musick Jr.

Abstract

Belief Network Induction

by

Charles Ronald Musick Jr.

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Stuart J. Russell, Chair

This dissertation describes BNI (Belief Network Inductor), a tool that automatically induces a belief network from a database. The fundamental thrust of this research program has been to provide a theoretically sound method of inducing a model from data, and performing inference over that model. Along with a solid grounding in probability theory, BNI has proven to be a quick, practical method of inducing data models that are highly accurate. The results include a belief network that stores beta distributions in the conditional probability tables, coupled with theorems demonstrating how to maintain these distributions through inference; techniques for applying neural network and other learning techniques to the task of conditional probability table learning; and a decision theoretic sampling theory which addresses scalability issues by characterizing the size of the sample needed to produce high quality inferences.

The setting for this work is in database mining. Database mining is one of the fastest growing topics in Artificial Intelligence today, with industry providing at least as much impetus as research labs and universities. The general goal is to extract interesting quantities or relationships that are "hidden" in large corporate or scientific databases, with the potential benefits of a successful technology being enormous. For example, models can be built that characterize what types of customers will respond to what types of marketing schemes, retailers will be able to predict sales to help determine correct inventory levels and distribution schedules, and insurance companies will be able to predict expected claim costs and better classify who will buy what type of coverage.

Acknowledgements

My time at Berkeley has been one of learning; my tutors have been my professors, my students, my friends, and my family. The results that are examined in this dissertation give just the barest hint of the the growth that has occurred.

In particular, I am deeply grateful to my advisor Stuart Russell. He has taught me how to think critically, and has instilled in me the ability to formalize knowledge on paper. Stuart has the uncanny ability to see to the core of any problem that is brought to his attention, which has made him a matchless source for new ideas. In the same way that Stuart has shown me how to be a researcher, Brian Harvey has shown me what it means to be a teacher. He has been an inspiration to every graduate student that has taught with him; the results of his efforts are evident not only in his students, but also in the students of the teachers that he has taught.

To my fellow RUGS (Russell's Unusual Graduate Students) members, you have been a constant source of stimulation, both intellectually and otherwise. It has been a long road, and I can not think of better travelling companions. Garyo, as my officemate for five years you have been the executioner of my bad ideas and the proponent of the good ones; you have also been an excellent friend. To my classmates throughout four years of Chinese, you have added the spark of the extraordinary to my life as a graduate student. Especially my tutor and wonderful friend Huanran, you turned that spark into a steady blaze.

My friends Mike, Jeff, Huanran, Garyo, Tony, Eden, Emy, Todd and Don, our countless discussions about anything and everything have helped me through the difficult portions of this journey, and have magnified the pleasures along the way. Thank you. Finally, I thank my family. You have always provided the support that makes anything seem possible, and the love that makes it all worthwhile. I dedicate my thesis to you.

Contents

List of Figures

List of Tables

Preface

Two of the chapters in this dissertation follow closely two previous publications. Chapter 3, entitled "Propagating Distributions", is similar to my paper "Maintaining Inference Distributions in Belief Nets"[63] which appeared in the conference on Uncertainty in Artificial Intelligence in 1993. Chapter 4 "Decision Support" is in large part a paper that I coauthored with Jason Catlett and my advisor Stuart Russell, "An Efficient Method for Constructing Approximate Decision Trees for Large Databases"[64] which appeared in the Machine Learning conference in 1993. In this paper, Jason provided the empirical evaluation, Stuart and I were responsible for the theoretical details.

Chapter 1

Introduction

1.1 Motivation

Information is useless without knowledge of how to use it.

This is the principle behind any system that attempts to build a model of some domain, then reason from that model instead of from the domain directly. Data in its raw form exists mainly to be processed. It is not organized to elicit any degree of understanding, it often contains a large percentage of random noise and irrelevant events, and it tends to be incomprehensible until summarized in some intelligent fashion. Consider the human senses. In order to make any sense of the stream of sensory input that we get from our eyes (millions of bits of information from rods and cones), our ears (two streams of multiband frequency readings), and our skin (billions of nerve pulses), the brain must first translate this information into a high level "language" that strips the noise and the majority of irrelevancies, presenting a clean, simple representation of the actual input. Without this high level model, most of the data would be completely unusable to us. Automatic model construction is one of the main objectives of Machine Learning, and is one of the fundamental problems in Artificial Intelligence. Automatic model induction is the topic of this dissertation.

Model induction is useful in a variety of settings, from both a practical and a theoretical standpoint, thus there are several possible modes of presentation of this type of research. This dissertation is presented in the context of database mining, which is the task of extracting the interesting and relevant information from a large database. Database mining is directly relevant to every computerized industry in the world, and it seems evident that the related technologies will end up being one of the most widespread and successful applications of Machine Learning technology that we have yet seen.

Databases are at the core of all but the smallest companies. They are the repository of any and all information that might be useful now or in the future to that company, and great troubles are taken in order to protect them from harm. But until recently, little has been done along the lines of *managing* all of the information stored in the database. How can we transform the raw data in a database into high level descriptions or models that present the useful information in easy to understand, easy to access forms? For years now corporate and scientific databases have been growing at tremendous rates, far surpassing our ability to manually dissect and analyze their contents. For example, Lawrence Livermore has over 32 terabytes of information representing the Lab's research data and results from the past fifty years. Other national laboratories, oil and gas exploration companies, and NASA all have databases that rival or exceed the amount of data stored at LLNL. The US Meteorological services [90] and NASA Jet Propulsion Labs are considering systems that can process one terabyte of satellite and telescope image data on a day to day basis. The National Storage Lab is working to build storage systems that can handle and provide access to quantities of data measured in pedabytes (a pedabyte is 1000 terabytes, or 1,000,000 gigabytes).

The result is that for the most part, the users of these large relational databases often have little idea of the type and value of the information that is contained in the raw data of their database. The pure size and complexity of these databases alone is enough to deflect any but the most serious attempt at modeling. The problem is compounded by restrictions that many companies place on the availability of certain sensitive, private or classified areas of the database.

Among the information that is hidden (or at least not directly represented) in these large relational databases is descriptions of the relationships between the variables in the database. Beyond indexing information, a relational database does not explicitly represent how, for example, the probability of an injury relates to the sturdiness of a car in a car accident, or what type of people have bought lawn mowers in the past five years. The goal of database mining is to bring all such relations to light in an easy to understand manner. The potential benefits of a successful technology are enormous. Some of the industries that will or have already been affected are [57]:

- **Marketing:** predicting which customers will respond to a mailing or buy a particular product; classifying customer demographics.
- **Banking:** forecasting levels of bad loans, modeling credit card spending patterns of customers to detect fraudulent credit card usage, and predicting which kinds of customers will respond to (and qualify for) new loan offers.
- **Manufacturing, sales, and retail:** predicting sales; determining correct inventory levels and distribution schedules among outlets.
- **Manufacturing and production:** predicting when to expect machinery failures; finding key factors that control optimization of manufacturing capacity; predicting excessive vibrations in a steel mill when rolling; determining values for circuit trim resistors.
- **Brokerage and securities trading:** predicting when bond prices will change; forecasting the range of stock fluctuation for particular issues and the overall market; determining when to trade stocks.
- **Insurance:** forecasting amount of claims and cost of medical coverage; identifying which factors have the largest affect on medical coverage; predicting which customers will buy new policies.
- **Computer hardware and software:** modeling disk drive failure; forecasting how long it will take to create new chips; catching potential security violations, and resource requirements.
- **Government and defense:** forecasting the cost of moving military equipment; testing strategies for potential military engagements; predicting resource consumption.

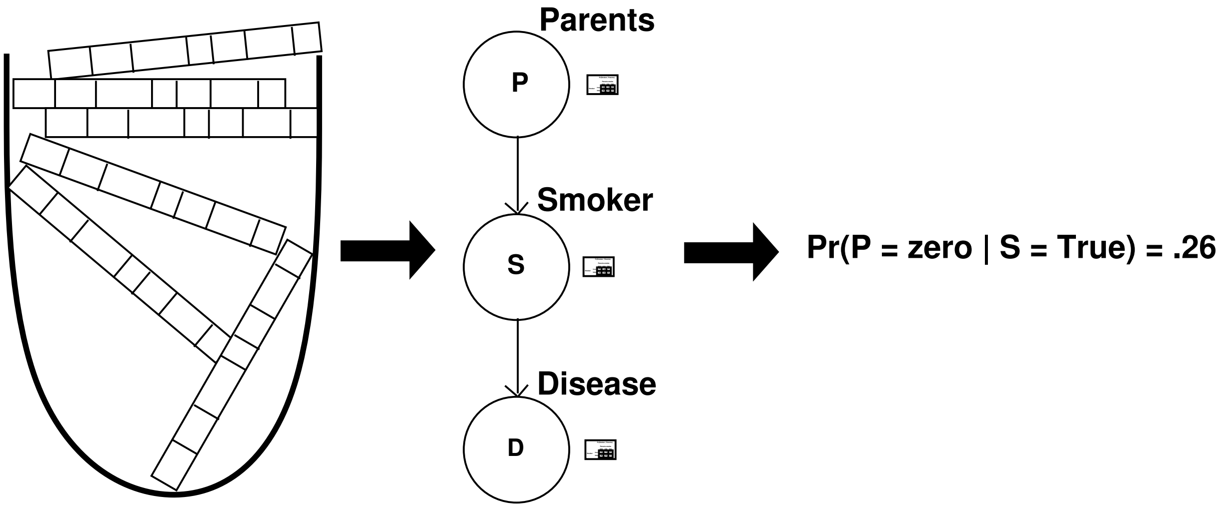


Figure 1.1: **Process Flow Chart**
The leftmost portion shows a database as an urn, indicating it to be a loose collection of datum that can be sampled from. The middle picture represents the model that will be learned from the database. The rightmost equation represents the desire to use to model to answer questions about the environment.

- **Medicine:** modeling a drug's mechanism of action; classifying anti-cancer agents tested in a drug screening program; allocating resources in emergency rooms; discovery of new cures.

This dissertation describes BNI (Belief Network Inducer), a tool that automatically induces belief networks from a database. The fundamental thrust of this research program has been to provide a theoretically sound method of inducing a model from data and performing inference over that model. Along with a solid grounding in probability theory, BNI has proven to be a quick, practical method of inducing data models that are substantially more accurate and complete than previous database mining techniques. Belief networks are the foundation of this approach since they allow the user to ask for any conditional or marginal probability in the network, they are incrementally updatable, and they provide a representation that is concise, understandable and amenable to manipulation consistent with probability theory.

Figure 1.1 shows a summary of the three main steps that BNI will carry out in describing or modeling the environment from the database; sample construction, model learning, and inference. BNI has made contributions to each of these tasks. The urn on the left makes the point that a database can be, and in many case *should* be sampled from instead of used in its entirety. The main question to answer here is how large the sample should be in order to build a model that is accurate enough to suit the user's needs. The condition "accurate enough" must be user-definable, as it will change depending on domain and situation. The middle figure represents the belief network model that is being learned from the data. The main goal in learning the model is to improve prediction accuracy over the environment. The righthand portion of the figure represents the desire to use the model to answer questions. The question BNI addresses in this portion is how to provide information that measures the degree of belief one can have in the answers returned by the induced model.

1.2 Domain Characteristics

What follows is a characterization of the domains that database mining techniques might be applied to. This is useful in identifying desiderata that should be satisfied by potential solutions.

- **Large amounts of data.** Today databases on the order of gigabytes are common; terabyte databases exist. Tomorrow, terabyte databases will be common, pedabyte databases will exist.

- **Sparse information.** Even a one thousand pedabyte database (10^{18} bytes) is not nearly large enough to characterize even a small fraction of the set of events that could occur in a moderate sized domain. For example, 100 variables with 5 values each gives rise to 5^{100} (or $\sim 10^{35}$) different events that could occur. Of course, it would be rare that all of the variables would be deemed as relevant or descriptive of the situations or events that are interesting. For example, the variable Car-Color is irrelevant to the expected cost of an accident; Car-Model would be much more relevant. Thus, all the data describing cars with different colors can be grouped together and treated as if the color variable does not exist, effectively reducing the number of "interesting" events. Continuing the above example, if only 10 of our 100 variables are needed to describe our most complicated interesting events, then there are on the order of 10^7 possibilities. A much smaller database can cover these events. However, even if just a few variables are needed to describe our interesting events, the fact that the events will not be equally probable leads to a gloomy conclusion; no matter how large or small the database, the event space will often be sparsely described. There will be interesting events that do not have enough information to characterize them.

- **Unfriendly data.** Continuous variables are possible. The data is expected to be imperfect, having both missing information, incorrect information, information that is uncertain by nature (the probability of heads when flipping a coin), and information that is not measurable but known to have an affect on the rest of the data.

- **Changing domain.** We live in a continuously changing environment, and must deal intelligently with a steady stream of incoming data. This data will be characterized by a certain amount of concept drift. The quantitative relations that exist during one time period are likely to change in the next.

These characteristics lead one to several conclusions. First, the model must be incrementally updatable. Updating is necessary because the model must be able to accommodate the steady stream of new incoming data. Incrementality is needed, otherwise the system will have to reprocess all previous data each time new data is received, and therefore will be hopelessly slow. Second, any practical system must be tractable in all phases from model construction to reasoning. The demand of tractability is especially harsh in this environment since the data set may be huge, and the user may require highly accurate models. Finally, the potential sparseness of relevant events indicates that the system should be robust and graceful in the presence of low probability events. All potential information available should be used when determining the probabilities of those events. Furthermore, since some events may be predicted with high accuracy, and others with very little, the system must be able to indicate a degree of confidence in the answers provided, otherwise it would be very difficult to have any degree of faith in the quality of the results provided.

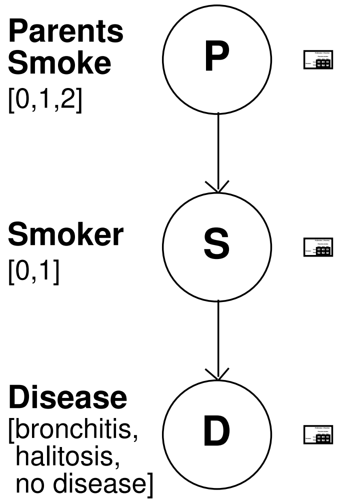


Figure 1.2: **A Smoker Belief Net**

A belief net showing a simplistic relation between smoking, bronchitis and having parents who smoke. The name of each node is in bold face, and the possible values appear under the name.

1.3 Approach

A quick description of the approach follows, then below each point is specifically considered. BNI constructs a modified belief network from databases that are replete with errors, uncertainty, missing and continuous values. The focus is on the induction and use of belief nets. We take advantage of the human ability to specify qualitative relationships and therefore make the assumption that the graphical structure is given. Given the graphical structure, we learn conditional probabilities as *distributions* instead of as point probabilities, and show how to propagate these distributions in inference. Where data in a node is sparse, we use traditional learning methods to supplement the statistical inference. To allow the learning techniques to scale up to large databases, we perform decision theoretic subsampling to decide the sample size that has the maximum utility. A hidden node is an unmeasurable variable in a database, a variable without any data. We add hidden nodes when the addition will dramatically reduce the size of the learning problem, and we discuss methods of supplying values to them.

1.3.1 Belief networks

The belief network is the model to be learned and used to reason with in BNI.

A belief network is a directed acyclic graph consisting of a set of nodes X_i for i in $1 \ldots n$, a conditional probability table T_i for each node, and a set of directed arcs between the nodes. Each node represents a random variable, and has an associated, possibly infinite domain $x_{ik} \in \mathcal{D}_{X_i}$. Note that in general uppercase will be used for variables and lowercase for values. An arc from X_i to X_j represents a dependency between the two, and establishes X_i as the *parent* of its *child* X_j. The conditional probability table (CPT) of every node in the network represents the set of

$Pr(P)$		
P	0	.62
	1	.23
	2	.15

$Pr(S\|P)$				
		P		
		0	1	2
S	0	.61	.57	.53
	1	.39	.43	.47

$Pr(D\|S)$			
		S	
		0	1
D	b	.75	.51
	h	.20	.41
	n	.05	.18

Table 1.1: **CPTs for the Smoker Network**

These are the conditional probability tables for the smoker network.

$Pr(D, P, S)$							
		P, S					
		0,0	0,1	1,0	1,1	2,0	2,1
D	b	.28	.13	.10	.05	.06	.04
	h	.08	.10	.03	.04	.02	.03
	n	.02	.04	.01	.02	0	.01

Table 1.2: **Joint Probability Table for the Smoker Network**

This is not a conditional probability table, but rather a joint table representing all possible events that could occur in the smoker domain.

conditional probabilities $Pr(X_i|\Pi_i)$[1], where Π_i represents the set of parents of X_i. Finally, X_i is conditionally independent of every other variable in the network given its parent instantiations, or $Pr(X_i|\Pi_i, X_j) = Pr(X_i|\Pi_i)$ for any $X_j \neq X_i$ and $X_j \notin \Pi_i$.

Figure 1.2 shows a simple belief net, with the domain of each variable printed beneath the name of the variable. The corresponding CPTs are shown in Table 1.1. Table 1.2 shows the joint probability table for the problem, where the joint table explicitly describes the probability of every possible combination of variable values. The joint is typically large enough even in smaller networks that it is not directly representable on current storage systems.

[1] This is actually shorthand for all probabilities matching $Pr(X_i = x_{ij}|\Pi_i = \pi_{ik})$

In fact one of the primary benefits of belief networks is space compaction. In Table 1.2 seventeen independent cells are required to represent the joint probability table, whereas only nine independent cells are required to store the belief network. The factorization gets much more dramatic if the number of parents grow, or the domain sizes grow. For example, if each node had ten possible values, then the joint would require about one thousand cells, while the belief net would require a little over two hundred. The space compaction is achieved by taking advantage of the known independence conditions between the variables in such a way that the belief network still accurately represents the probabilities stored in the joint space. For any belief network, the joint probability space can be reconstructed by multiplying through all the conditional probabilities represented in the network, as $Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} Pr(X_i|\Pi_i)$.

There are just a few other related definitions to cover. Let Ω and Ψ be mutually exclusive subsets of the nodes in the network. An inference problem can be defined as the task to provide a value for a query of the form:

$$Pr(\bigwedge_{X_i \in \Omega} X_i = x_{ik} \mid \bigwedge_{X_j \in \Psi} X_j = x_{jp}). \tag{1.1}$$

For example, the question $Pr(\text{Disease} = \text{bronchitis} \mid \text{Smoker} = \text{true})$ is an inference problem involving the Disease and Smoker variables in the network. Inference in a belief net can be done in accordance with the axioms of probability. In most cases it is reasonably quick, however the worst case is NP-complete. There are techniques to generate approximate results with user controlled bounds on the amount of time used, the most prevalent of these being stochastic simulation.

In this dissertation, *updating* refers to the process of computing the posterior probabilities in the belief network on the basis of a set of data. The idea is that when constructing the network, the CPTs are initially given a prior distribution, then these distributions are modified or updated with each additional sample seen. Belief networks are incrementally updatable, meaning that the network can be updated sequentially on the fly, as opposed to batch processing large sets of samples. The updating process is well understood [23, 47, 71], and we go over it in depth in Section 3.1.1.

1.3.2 Learning

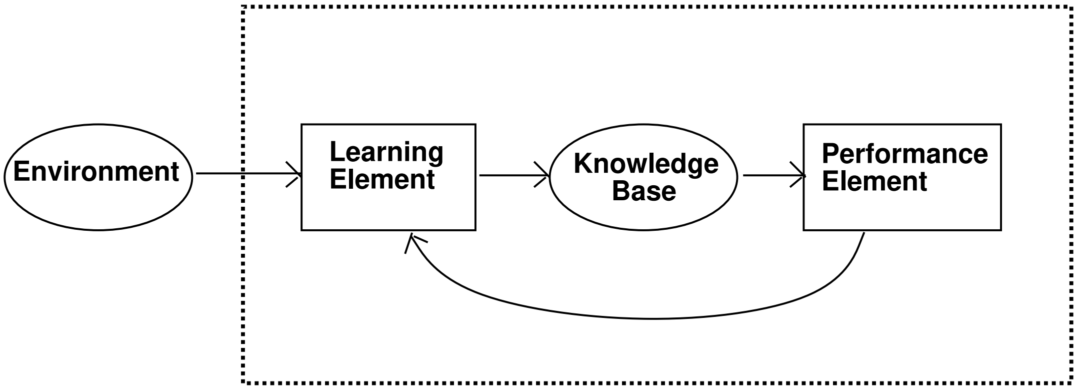


Figure 1.3: **A Model of Learning**

We want to learn about the real world through a database; to reason about and make predictions for the real world based on what has been discovered in the process of learning.

The form of learning that we concentrate on is the construction of a simplified model of the environment with the express goal of facilitating a question and answer process, where questions take the form of Equation 1.1. There is an abstract model of learning from Dietterich [28] that lends a bit more form to our description. The model consists of four elements, the *environment*, the *learning element*, the *knowledge base* and the *performance element*. The model is shown in Figure 1.3. The environment is the world, or the portion of the world that is being considered in the problem, while the knowledge base is a model of that environment. The learning element is the system that modifies the knowledge base on the basis of the environment and feedback from the performance element. The performance element uses the model to do inference.

The environment in BNI is seen through the database that we learn from, meaning that we are taking the database to be representative of the world. The knowledge base is basically a belief network representation of the database. The performance element consists of a fairly standard set of inference algorithms that produce probabilities over sets of variables in the network, plus a new theory that gives the power to produce probabilistic distributions as answers as well. Finally we have the learning element. This dissertation begins with simple statistical induction to learn the conditional probability tables of the belief network, and then works on improving the learning process to take better advantage of the available information.

The exact nature of the BNI learning task has been briefly touched on, and now that belief networks have been described we can go into more depth. Looking back, Figure 1.1 shows a summary of the three main steps that BNI will carry out in modeling and describing the environment from the database; sample construction, model learning, and inference. These correspond to selection of the environment, the construction and updating of the knowledge base, and the operation of the performance element.

When constructing the sample, the specific task is to randomly choose a subset of the database with the purpose of using that sample to learn a model of the environment. A significant part of the problem is to determine the size of the sample that is actually needed in order to induce a model that is good enough to solve the problem at hand. To date little thought has been given to this problem, meaning that no matter how large the database, and how much (or little) the user cares about accuracy, the entire database will be used for learning the model. This does not scale up very well in the face of slower learning techniques or large databases, and is simply not the economically rational approach in the time constrained environment in which we live.

Model induction is the second component of BNI, the task being to induce a belief network from the data in the sample. Belief networks were designed from the beginning to model probabilistic domains and to handle the incremental updating of the beliefs contained in the model. However, there have been notable gaps in the field concerning issues related to induction. Portions of this problem have received some attention recently [22, 23], dealing for the most part with learning the structure of independencies in the model, or learning the *qualitative* or *graphical* structure. The focus of the model induction component of BNI is instead on improving the current theory and practical methods for learning the *quantitative* structure of the network, or the conditional probabilities that make up the model.

This focus has been chosen to shore up the weak link in the model creation chain. As humans, our ability to make qualitative judgements about which variables affect the value or state of another is good, and this ability can be directly applied to establishing the graphical network structure. On the other hand, our ability to make quantitative statements is extremely poor [99]. For example, where we might be able to say that weather has an effect on traffic, when asked to predict numerically how the throughput on I880 will vary with snow, chances are that we would not be able to make an accurate guess. Whether we are constructing an expert system from a training set [48, 62, 10], discovering regularities in a scientific or business database [54, 55, 56, 49, 101, 104, 103], or more generally, building a belief network from a database [29, 42, 73], we can take advantage of our innate ability by specifying the qualitative structure of the database. But then, other methods must be used to aid our judgement in the specification of the quantitative structure. The typical approach is to learn the CPTs with statistical induction techniques, we expand this and show results of applying neural nets, decision trees and non-linear regression techniques to this problem.

The third component of BNI is the inference system. The main goal in this portion of the dissertation is to provide precise inference *distributions*. In a belief network, the learned probabilities are normally point probabilities, therefore the inference are typically also point probabilities. The problem with a point probability is that there is typically no process or information that can take two point probabilities and distinguish the one that was produced from a large set of data from the one that suffered from a lack of relevant data. Up until now there have been no methods with the capability of manipulating and providing precise inference distributions. Many systems simply do not consider this issue and just provide point probabilities as inference results. Others provide estimates of the Bayesian distributions, or instead decide to maintain bounds on the inference probabilities.

Statistical Induction

Let θ represent the unknown probability of some event occuring. The statistical induction approach being referred to is the task of estimating θ from a sequence of observations of the event that it describes. We delay an in-depth discussion on this topic until Section 3.1.1 which establishes the context necessary for understanding the rest of Chapter 3. For now it is enough to accept that the beta distribution is a fitting choice for representing θ. See Appendix A for an introduction to the beta distribution.

CPT induction is mapped to the statistical induction problem by taking each cell in a CPT to be an unknown conditional probability $\theta_{i,j,k}$ (the cell for node X_i, value x_{ij}, and parent instantiation π_{ik}), and to treat the database as a set of observations **S** of these probabilities. It follows then that each unknown $\theta_{i,j,k}$ will be estimated by a beta. This method of learning the CPTs is the strawman that we use to evaluate the strengths and weaknesses of our new learning approaches.

We have chosen to compare our methods to this strawman for two reasons. First, it is the standard learning mechanism for systems that induce belief nets. The more compelling reason is that when running an SQL query on a relational database, the events relevant to that query are returned. The counts of these events are sufficient to produce the beta distributions describing the events; this basically means that the standard relational database query techniques are equivalent to the strawman. The strawman then is the method currently in use in nearly every existing database.

One of the particular results of this type of learning is that each explicitly represented conditional probability $\theta_{i,j,k}$ is learned as a distribution, rather than as a point probability. There is often confusion as to what meaning this second order distribution (a distribution over a probability) could have, but the interpretation is simple, meaningful and natural. A distribution is a weighted range of possible values for some random variable. The heavier the weight is, the more likely that value is. If that random variable *happens to be* a probability, then distribution is actually a second order distribution. The fact that each $\theta_{i,j,k}$ is being learned on the basis of a set of observations of the real world implies that each is in essence a random variable, and therefore the beta distributions are actually second order distributions.

In general, whenever a model is constructed from observations, it is more advantageous to represent the learned quantities as distributions rather than point probabilities, or means. To begin with, the information in a distribution subsumes the mean in that the mean can be produced from the distribution if desired. The other information represented is useful as well, giving a good indication of the stability of the current estimate. This information has the potential to affect decision making in many ways. For example, a company working on particularly sensitive problems might only be willing to accept and use an inference if it can be made with a specific degree of confidence or certainty. A distribution provides the information necessary to compute these degrees of belief. More meaningful comparisons can also be made in situations where there are multiple models of some task with each model producing recommendations. For instance, instead of choosing the recommendation with the highest probability of success (maximize potential gain), it might be more desirable instead to choose the recommendation with the most certain information so as to minimizes potential loss.

Beyond Statistical learning

By using other learning algorithms, it is possible in many cases to do much better than the statistical approach. To understand why, we must first conceptualize the structure of the data, and how it relates to the CPTs. We use Figure 1.4 for reference. First, every table represents a set of conditional probabilities $Pr(X_i|\Pi_i)$. Assuming without loss of generality that the data in the database is complete (no missing values), then each sample will assign a value to each variable, and thus each sample will be relevant to every individual CPT in the belief network in some way. The

		.01	.01	.15	.8	.03
		Leaves on a clover				
		0	**1**	**2**	**3**	**4**
happy child	**false**	β(1,1)	β(1,1)	β(2,12)	β(65,15)	β(1,2)
	true	β(1,1)	β(1,1)	β(12,2)	β(15,65)	β(2,1)

Figure 1.4: **Conditional Probability Table for a Happy Child**
This table is Pr(happy child | Leaves on clover). The numbers at the top of the table represent the probability of that instantiation of parent variables.

unique parent instantiations for any node X_i are mutually exclusive, each one is a column in the CPT for X_i. Each sample will be relevant to exactly one column in the CPT, the column for which the parent instantiation is consistent with the sample. Thus, to describe the CPT, there must be enough samples to cover each column in each CPT.

In a realistic domain the number of unique parent instantiations for a node can be a large number. The situation is aggravated by the fact that the probability of each parent instantiation is not the same. What often happens then is that a table will have a small fraction of the parent instantiations that together have a very high probability, and thus take in the bulk of the data. Consider Figure 1.4, where the 2 and 3 leaf clovers represent 95% of the total number of clovers that can be found. The set of low probability columns (0, 1 and 4 leaf clovers) rarely see the light of day. The implication of this is that to cover a table, the number of samples already likely to be needed is multiplied by the inverse of the probability of the lowest probability column. The upshot of this is a technique that requires an overabundance of data and time to process the data, and produces inferences that are less accurate than they could be because they do not use all the information available.

The problem is that statistical induction assumes that the data in each column of a CPT is *independent* of the rest of the columns, and thus each column must be learned independently.

The approach taken here is to apply other learning techniques like neural networks, non-linear regression techniques, and decision trees to the task of learning the CPTs. These techniques all apply some bias to the problem, somehow combining information from all the columns in order to better learn each individual one. This has been the standard theme in induction ever since the weaknesses inherent in the idea of learning by memorizing every possible event (table learning) were exposed. It is interesting that this is also similar in concept to analogical [31, 81] and case based reasoning [72, 2]. They all take the approach that by supplementing direct experiences with knowledge derived from relevant but slightly different situations, the end product will be faster learning, and the ability to cope with situations previously unseen.

A primary task of BNI is to learn a good conditional probability table for each node. BNI is flexible enough to attach a different learning mechanism to each conditional probability table, learn each table according to the selected method, then put the results together in the final belief network model. We can go even further by mixing and matching methods, such that for columns in a CPT that are data rich, we ignore whatever learning function was specified for that table and just return the beta distribution derived from the statistical summary. These columns are then merged with their data-poor neighbors whose values have been supplied by the selected learning method. On the insurance database that we have used extensively for testing, this has resulted in 25% to 60% improvements in general on the accuracy of the learned tables.

1.4 Thesis statement

The BNI approach enhances the benefits of statistical rote learning techniques with new, practical theory for decision support and inference mechanisms, smoothly integrating these ideas with other learning algorithms such as neural nets or nonlinear regression to provide a cohesive outlook on the learning and use of belief networks for decision making and modeling. With this approach, we show the following:

Thesis:

The BNI approach provides a practical and effective theory for the automatic induction of useful models of large sets of data in large real world applications where the data may be erroneous, uncertain, and incomplete. Specifically, new results include:

1. Theorems demonstrating how to maintain probability *distributions* as opposed to point probabilities when computing probabilites from the network,

2. A decision theoretic theory for subsampling to determine the sample size required to induce a model that is accurate enough to produce inferences with specifiable degrees of accuracy.

3. Techniques for applying neural network, non-linear regression, and decision tree algorithms to the task of conditional probability table learning.

4. An implementation of the above demonstrating enhanced conditional probability tables that provide on average from 25% to 60% improvement over traditional statistical methods.

1.5 Outline

This dissertation proceeds as follows. Chapter 2 provides a description of previous work in related areas, and the impact it has had on this dissertation. Chapter 3 describes in full detail the statistical learning process, and lays out the theory for manipulating and returning inference distributions in the belief network. Chapter 4 describes the sampling theory that is useful in dealing with problems of scale, and user established requirements on accuracy. Chapter 5 describes the use of neural network, decision tree and non-linear regression techniques in learning CPTs. It also describes issues in learning hidden node, and details the results obtained from an implementation of the ideas. Chapter 6 is the conclusion.

Chapter 2

Previous Work

This chapter briefly discusses previous related work, setting the background for the development of belief network induction and related topics. This is not meant to be comprehensive as some of the areas that are discussed are too broad to be fully covered. This should be viewed as an introduction to related work and to different approaches to related problems.

2.1 Belief Network Learning

In AI there has been a relatively long history of induction [91], but a relatively short history of complex probabilistic model induction, particularly the induction of belief networks. This section presents some of this work, including the induction of both belief networks and influence diagrams (an influence diagram is a belief network with decision nodes, and utility nodes added).

There are three aspects of belief network learning that we consider independently in order to add some structure to this presentation, though in reality the boundaries between the three are often indistinct. The three sub-problems are the learning of the graphical structure of the belief network, the learning of the conditional probabilities, and the inference processes in the belief network. The inference processes are relevant, as the types of inference algorithms that can be applied will be dependent on the type of learning that has been done.

2.1.1 Learning Structure

This dissertation makes the assumption that the graphical structure of the belief network has been provided already, either by knowledge engineers, domain experts, or an automated structure discovery algorithm. One of the main reasons for this assumption is that humans tend to make much more accurate qualitative statements than quantitative, implying that human supplied structures tend to be fairly accurate.

Another reason is that there are already several methods that are well on the way to doing the right thing when it comes to discovering graphical structures. In particular, Cooper and Herskovits [22] have presented a Bayesian method for doing a greedy search among possible network structures to try to find the most likely structure. It is not clear how well this approach scales up to large networks, but the ideas are very appealing and the general mechanism seems correct.

The goal is to find the most probable belief structure B_s given a database D of instances, or to maximize $Pr(B_s|D)$. This task is very difficult, as the number of possible structures is super-exponential in the number of variables, and there is no clear way of effectively pruning the search space as synergistic affects might occur between any set of variables. Five assumptions are made, namely that

1. The process that generated the database is representable by a belief network containing just the variables in B_s, which are discrete.

2. Cases occur independently, given a belief network model.

3. Cases are complete, that is, there are no cases that have variables with missing values.

4. The belief about the assignment of a value to a conditional probability is independent of the assignment of a separate conditional distribution. For example, $Pr(X_3 = 0|X_2 = 0)$ is independent of $Pr(X_2 = 0|X_1 = 0)$.

5. The density function $f(B_p|B_s)$ is uniform, where B_p is the quantitative structure, or the probabilities. This says that there is no initial preference as to what probabilities to place on the structure B_s.

Assumptions 1 and 3 are removed in later papers [23, 21] where they show how to deal probabilistically with hidden variables and missing values (see also York and Madigan [102]). The last assumption is relaxed in the same paper to allow a Dirichlet prior. From Bayes rule, we know that the probability of a structure given a database of examples D is found as $Pr(B_s|D) = \frac{Pr(B_s,D)}{Pr(D)}$. If instead we have a pair of structures, and we need to choose the most probable of the two, then we can use the relative probabilities

$$\frac{Pr(B_{s_i}|D)}{Pr(B_{s_j}|D)} = \frac{Pr(B_{s_i}, D)}{Pr(B_{s_j}, D)},$$

obviating the need for a prior on the data. Cooper and Herskovits therefore use the above assumptions and Bayes rule and derive an expression for $Pr(B_s, D)$. The algorithm starts with a one-node network, and repeats the following steps.

1. Add a node to the set of parents of the current node.

2. Calculate the probability of the new structure and compare it to the old.

3. If the new structure is noticeably better, then keep the newly added node and try another.

They make the point that this is just one of the many potential algorithms that are possible in tackling this problem.

Given the size and the complex nature of the problem, their results are surprisingly good. The algorithm can reproduce models that are nearly equivalent to those selected by humans on small and medium sized problems. Also, with an approach of this nature it is clear that this is a good interactive modeling tool, able to give the user feedback on the probability that adding that arc will actually improve the network.

Finally, in deriving closed forms for $Pr(B_s|D)$, Cooper and Herskovits show expressions for the conditional probabilities in the CPTs based on frequency counts obtained from the data. These frequencies, or cell counts, are later equated to the sufficient statistics of the beta distributions[1] that were described in Chapter 1. The sufficient statistics of a distribution are the set of parameters that are necessary to describe a distribution in detail (for example, a Gaussian needs the mean and standard deviation). The beta distribution turns out to be fundamental to our method of manipulating the inference distributions, and we will explain its origin in depth in Section 3.1.1.

There is other work that complements this approach rather well. Part of the process of building a relational database is to put it in BCNF form [19, 18, 20], or to *normalize* it. The goal of normalization is to minimize the duplication of stored information and to speed search times by properly structuring the functional dependencies within the tables. Wen [98] shows how to map the results of the normalization process into a belief network structure. It is easy to imagine this being used as a starting point for the Cooper and Herskovits algorithm mentioned above.

There has also been work in the area of belief network reformulation, where the goal is to be able to sensibly modify the current structure of the belief network to explore new ideas, or try to improve the modeling performance of the current structure. When building a belief network, one decision that must often be made is how to discretize the continuous variables in order to fit them into the standard belief network model. This discretization problem is also termed the *granularity* problem. Chang and Fung [14] tackle the granularity problem by making it easier to visualize and explore the space of different granularities, and to help the process of choosing a proper discretization of continuous variables. They present a fairly user-intensive refinement process whereby the user provides a mapping from an original set of node values for one node in a network to new set. The mapping is a *refinement* if the new node is larger (by adding more node values we are looking at the distribution in more detail), and is a *coarsening* if the new node is smaller. Their task then is to find new probabilities for the CPT of the node such that the constraints set upon the values from the old network are not violated. They maintain locality, isolating the changes from the rest of the network by working entirely within the Markov blanket of the node in question. The Markov blanket of node X_i consists of the parents and children of X_i, and the parents of the children of X_i.

There are many other approaches to graphical model construction [11, 7, 24, 43] that we will not explore in this dissertation.

2.1.2 Learning Probabilities

In this section we examine the different possibilities that exist for estimating the conditional probabilities.

When constructing a network, the most common mechanism for updating the conditional probabilities is to do simple bookkeeping and count where the samples are *hitting*. For the conditional probability $\theta_{i,j,k}$ ($Pr(x_{ij}|\pi_{ik})$), a hit occurs when the sample instantiates the variables X_i and Π_i as x_{ij} and π_{ik}. Doing this gives a simple mechanism for generating point probabilities that are consistent with the data, as well as supplying the sufficient statistics for several representative distributions.

[1] Actually, Cooper and Herskovits recognize the statistics as those of the Dirichlet distribution, the n-dimensional beta distribution.

There are other ways to learn the conditional probabilities; in many cases these methods far outperform the bookkeeping described above. For example, this dissertation describes a way of applying neural networks to the task of learning CPTs. Neural networks have been used by others as well.

The work that bears the closest resemblance to this dissertation was developed independently in Tseng's thesis [97]. The goal behind combining neural networks and belief networks into a single model is to use the strengths of each approach in a way that accents the power of the overall model. The advantage of the neural net is that it is a powerful learning mechanism with the potential for a fast parallel implementation. Among the benefits of belief networks, the chief advantage is its ability to break the overall modeling task down into local components that then can be modeled with neural networks. The complexity of each local task is much less than learning the overall domain, thus further increasing the efficiency and effectiveness of the neural network approach. Also, by mapping the learning results back into the belief network, we are not limited to answering one question or learning one function (as the case with a neural network), but instead can answer any general query.

Tseng actually uses an influence diagram to structure the learning process. Making the assumption that each underlying set of conditional probabilities has a multi-modal normal distribution, he constructs a specialized Self Organizing Probabilistic Neural Network (SOPNN) that essentially combines normal distributions to build a model of the conditional probabilities. The idea is similar to the fact that any continuously differentiable function can be represented by a family of sine and cosine functions.

The presentation of the SOPNN model is very detailed and mathematically rigorous. Tseng proposes a rather unique neural network construction and specifies algorithms for setting the weights, updating the neurons, and propagating the evidence. We do not describe the construction in great detail as it has minimal impact on this dissertation. At a high level, an SOPNN is built for each family of nodes in the influence diagram, where a family is a node as well as all of the parents and all of the children of that node. It is not clear how Tseng integrates the SOPNN for X_i and the SOPNN for a child or parent of X_i, indeed it seems that he basically considered the task to be to model only one family within each influence diagram.

Consider the problem of learning a CPT. Each column of the CPT has been assumed to be a multi-modal normal distribution. The main feature of an SOPNN is that it proposes to model the multi-modal normals with K localized Gaussian receptive fields, where K is a user-specified parameter. The construction of the K receptors gives a good amount of insight into the workings of SOPNN. The purpose of each of the receptors is to represent a cluster of the input data with a normal distribution (the distributions are later added together and normalized to produce the output). The standard deviation of the normal is user-specified and is the same for each receptor. The mean of the normal is equivalent to the mean of the data in the cluster. The main difficulty with the construction of the receptors is that a greedy approach is used to determine the clustering of the data. Let each sample s_i be a vector consisting of the node value x_{ij} along with values for the parents of X_i.

1. Construct the vector X^c for each cluster C, where X^c is the vector mean of the data in the cluster.

2. Get a new sample s_k, attach the sample to the cluster with the lowest mean-squared error: $(X^c - s_k)^T(X^c - s_k)$.

3. Goto step 1.

It is not clear in the thesis how the initial X^c vectors are established. The two possibilities that seem consistent with his approach are that the initial vectors are user-specified, or they are randomly set from the first K samples in the data, 1 sample per receptor. What is clear is that by clustering the data like this, the output of SOPNN will be strongly dependent on the order of the input data. In particular, a poor spread of the initial K samples will result in poor clustering for the receptors, adversely affecting the results.

Tseng's thesis shows impressive results for cases when the underlying distributions are normal. Since SOPNN does not need a negative training phase, it is faster than a Boltzman machine (the typical neural network construction for generating probability distributions; see [66] for other work in this area). There are, however, some difficulties in interpreting the results, as the method of testing was not discussed. It is not clear how much data is being used, how many times each test case was run, how the input data was generated and organized, how large K was chosen to be, and how the cluster vectors X^c are initially chosen. The approach in general has a few weaknesses as well. The biggest concern is that it is not at all clear what will happen to performance when non-Gaussian distributions need to be learned. As discussed, the clustering method used leads to a strong dependence on the input order of the data, and also makes it impossible to model nominal variables. It is not clear what a good choice of K will be, but it is clear that the best choice will be different for each SOPNN. Finally, the rigidity of the construction of SOPNN and its integration into the influence diagram is overly restrictive, as there are many situations in which it will be more appropriate to use other learning techniques, such as decision trees, more general neural nets, or regression techniques.

A different viewpoint on learning the probabilities comes from Madigan et. al. [59]. They use the bookkeeping approach to generate probability values for the CPTs in each belief network, but with a very interesting twist. Their premise is that when inducing belief networks, it is incorrect (or at least inaccurate) to propose a structure with a (relatively) high probability given the data, and then use only that structure to determine the probabilities for the model. Their approach is to take the structural uncertainty of the model into account when determining the probabilities in the model. For example, instead of returning the $Pr(x_{ij}|\pi_{ik})$ of the most likely model, they return the average of that probability over all of the high probability models. Their distribution over the model space is based on Occam's Razor; simpler models that are as descriptive as more complicated models have a higher probability. Neuenschwander and Flurry [68] confirm this preference system in a way, showing that a higher model complexity carries with it a larger degree of instability, implying that more data would be needed in order to justify the use of a more complex model.

A similar approach comes from Paab [71], but where Madigan actually precomputes and stores the new probabilities, Paab delays the computation until inference is performed. He defines a set of possible worlds (analogous to Madigan's multiple models), then uses these possible worlds to aid in the computation of the probability at hand. Specifically, if the goal is to generate some confidence that the desired probability is within a certain interval, this can be calculated by summing over all the possible worlds whose probabilities lie within the interval. Of course, something like this would be impossibly slow if one were to actually try it, but approximations can be made, and the presentation of the calculation of the possible worlds is nice in itself.

The rest of this section deals with learning second order distributions or intervals as opposed to point probabilities. The main purpose behind the second order distributions here is to

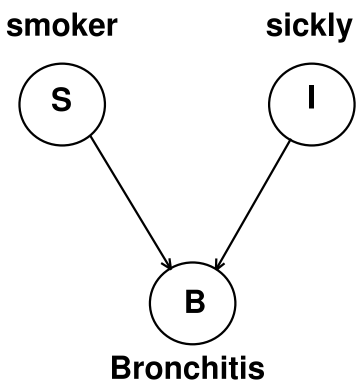


Figure 2.1: **A Bronchitis Belief Net**
This shows a model of the dependency of bronchitis on whether the patient is sickly, and whether the patient smokes.

provide a way to determine some degree of belief in the learned probabilities.

The approach taken by Spiegelhalter and Lauritzen [94, 93] is to specify two types of variables in the network: *core* variables, and *uncertainty* variables. The core variables correspond to typical belief net variables. Each node has a set of core variable parents, and a set of uncertainty variable parents (one for each unique instantiation of the core variable parents). The uncertainty variables provide a discrete specification of the second order distribution of the corresponding conditional probabilities. Spiegelhalter and Lauritzen show how to update these uncertainty variables, and mention their use in providing estimates of the inference distributions as inference is done. The selling point of this approach is the smooth integration of the uncertainty variables into a speedy inference algorithm. On the down side, there are three factors that degrade the quality of these distributions. First, the addition of the uncertainty variables to the network may substantially increase the size of the representation that is constructed in that inference algorithm. Second, as with most statistical approaches, a strong independence assumption is made which they call local independence. The assumption is that the uncertainty variables attached to a particular core variable are mutually independent. As an example of what this implies, from Figure 2.1 we could say that the distribution of $Pr(bronchitis \mid sickly, smokes)$ is independent of the distribution of $Pr(bronchitis \mid \overline{sickly}, smokes)$. The last factor is that the uncertainty variables must approximate continuous distribution functions with discrete representations.

Klieter [47] starts in a similar fashion to Spiegelhalter and Lauritzen by transforming human supplied data into corresponding beta distributions. Then, for diagnostic inferences in a tree-structured belief network (actually, an expert system), an expression for the distribution of the inference is produced. This expression is approximated as a beta distribution, where the mean and variance for the beta are produced from approximations of the mean and variance of the original expression. This approach, though complicated, yields good results for the particular inference problem that is analyzed. It would require more work to extend the results to inference within general DAG-structured belief networks. Neapolitan and Kenevan [67] have done similar work, concentrating more on bounding the *variance* of the distribution while doing inference. Their approach require a tremendous amount of work and space; it is necessary for the user to specify $E(p), E(p^2)$ and $E(p_1p_2)$ for every pair of probabilities in the network. Also, again it is limited to networks with tree structures.

Fertig and Breese [33] propose an approach based on interval probabilities. Using the fact that there is a set of network transformations with which any inference can be done in a influence diagram (a proof of this is alluded to in Olmstead's thesis [70] and a sketch of it is given in Chapter 3), they show how to keep consistent upper and lower bounds on inference probabilities. Unfortunately, the bounds tend to get weaker with each transformation, and intervals are not as informative as actual distributions. The concept of there being a small set of transformations sufficient to describe any inference in a belief net is, however, central to Chapter 3 of this dissertation.

2.1.3 Inference

We have talked about different approaches to learning the different parts in a belief network, now we discuss inference methods.

There are two schools of thought (or maybe one and a half). Inference in a belief network is the process of trying to find some conditional probability over two sets of mutually exclusive variables in the network. Several mechanisms have been developed to perform this inference, both exact and approximate. The main difficulty is that both the exact inference problem (Shwe and Cooper [89]) and the approximation problem (Dagum and Luby [25]) have been shown to be NP-complete. Dagum and Luby, however, have shown that for major classes of the problem, the task is merely polynomial in the size of the input. The goal is to find a technique that works well in most cases, where "working well" must be measured both by execution time and accuracy.

One form of exact inference is to use the set of transformations identified by Olmstead [70] and Shachter [86]. There is a minimal set of four transformations that are powerful enough to compute any inference in a belief network. These transformations can be precisely and simply defined, they are nice tools for network reformulation, and they tend to be very useful for theoretical work in a belief network. Chapter 3 goes into these transformations in much depth. Unfortunately, direct application is too slow to be an effective inference algorithm. The eventual hope is to be able to map these transformations into one of speedier inference algorithms below.

Henrion [41] shows several basic algorithms for speeding up exact inference. The basis of the algorithms is that in special cases of belief networks called polytrees for which there are O(n) algorithms for inference. If the network can be modified to be a polytree either by reformulation, or instantiating certain critical variables in the network (thereby removing that node from the network and leaving a polytree behind), then the fast O(n) algorithms can be applied. Another rather famous set of algorithms comes from QMR [42], where the specific structure of the belief network allows for specialized speedy "exact" inference algorithms. Breese and Horvitz [8] have analytically described the run-time tradeoff between the cost of reformulating the network compared to the expected required inference time if the current network is used, thus clarifying how and when reformulation should be done.

The other main category of exact inference is based on junction trees [92, 45, 51, 69]. The idea in this approach is to precompute much of the information in the network, and organize it in such a way as to minimize the need to do expensive work at inference time. The process is described in any of the papers above. A central concept to the approach is the *belief universe*. A belief universe is a clique of a subset of the nodes in the belief network. The belief universe stores the joint probability "mass" of the nodes in the universe, where the probability mass can be converted into conditional probabilities by normalizing to 1. The process then is to modify the current network so that a polytree of belief universes is created. When an inference is done, or evidence must be propagated, changes are made locally within the belief universe, and then the result is passed off to the next belief universe in the set. The main difficulty is that the size of the representation will be exponential in the number of variables in the largest clique, and the problem of finding the most parsimonious way to break the original networks into a junction tree is NP-complete. Even so, this approach is probably the most successful exact algorithm to date for belief networks.

One last note: Lauritzen [51] shows how to integrate the junction tree approach with methods of handling continuous variables. The methods rely on a very specific hierarchy and set of rules regarding the structure of the network and the relation between the continuous and discrete variables. It also assumes that the continuous variables are Gaussian in nature.

There are many forms of approximate inference. For example, Shimony and Charniak [88] translate a belief network into a network of Boolean variables, then give algorithms for working in this network to upper bound the aposteriori value of the nodes. The most commonplace by far, however, are techniques based on Monte Carlo sampling, Gibbs sampling, or logic sampling. The basis of these techniques is to use the belief network as a generator of random samples, check how many times the desired cases show up in the random sample, and from that compute the probabilities of those cases. There are ways of speeding this process up by selecting which sample to create and then discounting the value of the sample. These techniques are very necessary for doing fast inference, but are not strongly connected to this dissertation. For reference, see [87, 96, 39, 16, 89].

2.2 Database Mining

This dissertation concentrates on automatically inducing graphically appealing probabilistic models from data, and using these models to make predictions and reason about the environment. The field of database mining has not seen much work yet with these explicit goals, as there has been much more effort on the task of rule learning, and more generally function learning within the data. The quantity returned from a typical database mining approach is a list of rules of different strengths, where the rules describe simple mathematical functions between subsets of variables in the database. This type of result is different from what this dissertation is proposing for several reasons; primarily, the rules do not form a decent model of the data, and the general process is much less automated than that of BNI.

This work is interesting for several reasons. It would be expected in a scientific database that many of the quantities to be learned will have linear or simple non-linear relations between variables in the database. One would think that approaches that concentrate on learning these types of relationships will be able to find them faster and more accurately than other techniques such as the statistical induction described in Chapter 1, neural nets, or decision trees. We would like to be able integrate these particular function learning approaches into BNI in order to take advantage of any domain specific information, or representational advantages. For example, if the relationship between the parent variables and the node variable of a CPT is a linear function, then instead of the extensional representation of the CPT as a table filled with the conditional probabilities, an intensional representation that represents the CPT as a mathematical equation makes much more sense. This section explores some of the different approaches taken so far.

We look at rule-learning tasks first. A simple example of this task comes from the KiD3 project [75] in which Piatetsky-Shapiro presents a model of discovery where the learned rules are of the form $cond(A_i) \rightarrow cond(A_j)$. His approach is to let the user select the potential rule attributes A_i and A_j. Then the instances are hashed according to their A_i value and presented to the user in a modified form. The user is then asked to draw conclusions from this data.

Much of the value of the KiD3 work was actually in setting the field of database mining apart as an interesting and useful problem, along with identifying the goals to be achieved and the problems that must be overcome. As a general approach to database mining, KiD3 has some obvious difficulties. KiD3 built a hash table containing all the instances in the database. For a large database, this structure would not fit in the main memory plus swap space of any conventional system. The general lack of autonomy was bothersome, forcing a large degree of user interaction. The antecedent of the learned rule is too simple, the issue of uncertain data is not addressed, and it is not clear how to efficiently extend the hashing approach to more complicated antecedents. Piatetsky-Shapiro recognized several of these difficulties in the paper. He discusses the use of random sampling as a tool in managing the complexity of his approach. He also presents an analysis of the affect random sampling will have on the expected accuracy of the learned rules, given that the consequent is of the form $A_i = a$ or $a_0 < A_i < a_1$. This analysis is based on an urn model from Feller [32].

Later Piatetsky-Shapiro and Matheus [76] move to learning a form of determinations [81], which they call p-deps. They develop an interestingness function that removes a large portion of noise from the set of possibilities, relaxing the need for user interaction.

One interesting new approach from Matheus and Piatetsky-Shapiro [77, 60] is the claim that the truly interesting events in a database are not the discovery of existing rules, but rather the discovery of how these rules and quantities are changing as the database evolves (the discovery of deviations). The system KEFIR works in the health care domain, trying to answer questions such as "Why are GTE's health care costs rising?" and "What are the main factors in this rising cost?". They make use of existing domain knowledge on trends to pinpoint when trends are in line with overall industry trends, or are off track. With this data, a simple trend analysis can be done to determine that an abnormality exists, and by adding some knowledge of utilities, they can relate these changes to actual company costs.

The main difficulties that rule-based learning systems face are the impossibly large number of potential rules, and the need for domain-dependent knowledge to sensibly determine which rules are interesting.

The function learning approaches [101, 104, 103] that attempt to learn linear or non-linear functions within the data can be characterized by the BACON project. This research project actually covers a set of six versions of Bacon [49].

Bacon takes as input data for two variables X_i and X_j. In order to determine the relationship that exists between the two (if any), four heuristics are used to search the hypothesis space. One proposes X_j as a constant, one looks for a linear relationship between X_i and X_j, the other two are constructive induction rules forming new terms X_iX_j or X_i/X_j depending on the behavior of X_j as X_i increases. More extensive approaches were developed in the later versions of Bacon. The early Bacons were limited to learning relations between two variables, but later systems like Zytkow's Forty-Niner and Wu's KEPLER [101, 104] introduce some interesting ways to learn more complex relations.

Another look at a function learning approach comes from Frawley [36]. Frawley defines a new type of decision tree called a functional decision tree, and uses this to induce functions. The idea is that in a normal decision tree, it is not possible to represent relations like z = x + y. In fact, the set of relations that can be learned is rather limited. In Frawley's model, each node in the tree (each building block) can either be an attribute or a *function* of those attributes. Once functions are learned, they can then be interactively thrown back into the learning process.

Function discovery is a hard problem. Databases can be large, and the number of potential functions is infinite. The introduction of error into the system gives license to postulate almost any function as fitting the data. The main problem in these initial attempts at function learning seems to be the lack of communication between the AI and Statistics community. Statistics has been studying the determination of functional models from data for centuries, AI perhaps for two decades. Current AI techniques do not address issues of scale, examine a limited range of functions, do not deal convincingly with uncertainty or error in data, and have no sense of how to choose the "best" function out of the set of possibilities. None of the approaches from AI come close to current statistical packages such as LEAPS [38] that by linear regression can find all sorts of linear models quickly, and present rankings of the discovered functions based on MSE, Cp or Sp criteria. For more complex learning problems, packages for non-linear regression techniques such as PIMPLE [9] or MARS [37] are quite sophisticated, and produce good results. This dissertation allows any of these learning techniques to be inserted into the general belief network induction scheme as methods to learn the individual CPTs.

2.2.1 Clustering

This last section discusses clustering techniques. The goal of a clustering system is to take a set of instances as input, and output a set of groupings that reflect the natural clustering of the input instances. Although these techniques have little bearing on the methods used in this dissertation, they are a viable approach to database mining that are currently in use. For example, the problem of customer modeling (from the list in Section 1.1) is a natural application of these techniques. In a direct mailing scheme, a company will send out mailers to some population from its customer database. The customers that respond are modeled with these clustering approaches, and that model is applied to the entire customer database. The potential customers that match the model are then the targets of the second direct mailing. American Express has used techniques like this going from about 4% success rates up to about 10%.

Classification systems can be split into two categories, those that classify based on statistical methods, and those that classify based on conceptual coherency.

Research in numerical taxonomy and cluster analysis [3] instigated the Statistical Clustering class of techniques. The most current example is AutoClass II [17], a Bayesian classifier. AutoClass determines the number of classes to split the data into by using a hypothesis preference function that is a combination of Bayes theorem and the prior probabilities of the hypothesis. The result of classification is a taxonomy in which each class is described by the probabilistic distributions of the features. A taxonomy of this type allows the computation of the membership probability of an instance in a class, and the prediction of subsets of feature values for any given instance. Other nice features are that uncertainty handling and incrementality naturally fall out of the approach. Also, the user is insulated from the decision process.

AutoClass has some difficult problems. First, it is computationally expensive to compute the marginals, so iterative techniques have to be used to estimate them. Second, the learned taxonomy indicates that all attributes are important in predicting any other attribute; this is essentially equivalent to proposing a model that uses all the predictors available no matter what their significance. Finally, the biggest problem is that attributes are considered independent and normally distributed to make the approach computationally tractable. There are a few domains in which this assumption doesn't hurt too much, but in many it induces a high error rate. Wu et. al. [100] have tried to lower the error rate by using domain knowledge to classify examples whenever possible, thereby reducing the dependence on the Bayesian classifier. The idea was nice, but the results were a bit depressing, improving the error rate to 37% from 40%.

The second approach is Conceptual Clustering. The primary difference in this research compared to the statistical techniques is in the hypothesis preference functions that are used. The goal is to include preference functions that discriminate based on how well the instances match a predefined concept. This makes it possible to define domain specific preference functions. Michalski and Stepp [95, 61] describe Cluster/2 and Cluster/S in which the instances are first clustered into groups that maximize the clustering preference criterion, then a classification phase is initiated within each cluster. These programs use LEF (Lexicographical Evaluation Functional with tolerances) as the preference function. LEF is a list of user specified preference functions that are applied to the set of potential hypotheses. The low scoring hypotheses are discarded as each successive preference function is checked, until only one is left.

In [35, 34], Fisher et. al. take a more restrictive approach to conceptual clustering by defining what the "correct" preference function is. Specifically, they use category utility (CU). Category utility rewards classifications that both maximize $Pr(C_k \mid A_i = V_{ij})$ (the probability that the attribute value determines the cluster) and $Pr(A_i = V_{ij} \mid C_k)$ (the probability that the cluster determines the attribute value) over all the attributes. For this dissertation, finding classifications that allow high probability predictions of attribute values would be sufficient, so this preference function might be interesting to explore.

Out of a sense of completeness, we are compelled to address the origins of discovery, as database mining can easily be seen as a child of those first few systems.

Discovery in AI is often associated with Lenat's work on AM and EURISKO [54, 55, 56]. Other original work in discovery includes the projects GLAUBER, STAHL, DALTON, and DENDRAL [50, 10] which provide different ways to view discovery as a search through a hypothesis space, and DeJong's explanatory schema acquisition [27] which uses background knowledge to guide the discovery process instead of the more typical correlational techniques.

2.3 Sampling Theory

How much information do we need to make good decisions? Consider a manufacturing problem, where the variable X in question is the percentage of defective products coming off the assembly line. The possible actions are to accept the entire batch, or to test every element in the batch for defects and accept those that pass. The catch is that if the first action is chosen, but then it is discovered that there were more than $Y\%$ defects in the accepted batch, then it would have been more worthwhile to choose the second action on the assembly floor. The fundamental problem then is how much sampling should be done on the assembly floor in order to determine X

accurately enough to maximize the utility of the action choice.

In general the two-action decision problem is a fairly common phenomenon, and there has been a good amount of work done on this topic in the medical, business and scientific fields. A two-action decision problem is one where we have some information on two random variables X_1 and X_2 and can sample to get more. These represent, say, the chance of success of two actions that can be taken. There is a utility attached to each, which is commonly a function of the random variables. The core of the problem is such that with the information currently available on X_1, that associated action might be the correct choice to maximize utility, whereas if more information is gained, then the second action might become more attractive. The goal is to trade off the cost of obtaining this extra information against its value.

This problem was discussed by Raiffa [80], and then seven years later in 1968 Bechhofer et. al. [6] redefined it, established several variants and provided a first crack at a solution to some of the variants.

The problem is typically stated as the following. Let $\mathcal{N}_1, \ldots, \mathcal{N}_k$ for $k \geq 2$ be normal distributions with a known common variance σ^2 and unknown means $\mu_1, \ldots, \mu_k$. Rank the means μ_i as $\mu_{[1]} \leq \ldots \leq \mu_{[k]}$. There is a database with k attributes, each of these attributes has an associated normal distribution in $\mathcal{N}_1, \ldots, \mathcal{N}_k$. Find the attribute that corresponds to $\mu_{[k]}$, the distribution with the highest mean. The solution procedure $\mathcal{P}$ has to have the property:

$$Pr_\mu(CS \text{ using } \mathcal{P}) \geq P^*$$
$$\text{for all } \mu = (\mu_1, \ldots, \mu_k) \text{ satisfying } \mu_{[k]} - \mu_{[k-1]} \geq \delta^* \tag{2.1}$$

where CS is the correct solution and δ^* and P^* are user-specified parameters. This goal then requires a solution procedure to have a probability of P^* of correctly selecting the attribute with the highest mean, as long as there is at least a δ^* difference between the highest and next highest means. The format of the original problem Bechhofer et. al. stated was actually to set k to 2. The variants include allowing $k \geq 2$ as stated above, allowing the set of distributions $\mathcal{N}_1, \ldots, \mathcal{N}_k$ to belong to different classes of distributions, and finally allowing the procedure $\mathcal{P}$ to involve sequential sampling techniques.

The problem is a difficult one, and in general there is no normative method to date of generating the optimal sequential sampling strategy. We show a recent approach from Liu [58] that is fairly representative of the work that has been done on this problem for the past thirty years in this area.

The method of Liu [58] is, at stage n, for preassigned numbers $P^* \in (\frac{1}{k}, 1)$ and $\delta^* > 0$,

1. Take an instance $\mathbf{x}_{in}$ from the database, and rank it so that $x_{[1]n} \leq x_{[2]n} \leq \ldots \leq x_{[k]n}$.

2. Add the vector $\mathbf{x}_{in}$ to $\mathbf{y}_{in}$, which represents the sum of the n samples seen to this point.

3. Rank $\mathbf{y}_{in}$ so that $y_{[1]n} \leq y_{[2]n} \leq \ldots \leq y_{[k]n}$.

4. Eliminate as unable to affect the final outcome the $k_n - l$ attributes such that:

$$l = \min\left\{ q : 1 \leq q \leq k_n . \quad \frac{\sum_{j=k_n-q+1}^{k_n} e^{\delta^* y_{[j]1}/\sigma^2}}{\sum_{j=1}^{k_n} e^{\delta^* y_{[j]1}/\sigma^2}} \geq P^{*\frac{k_n-q}{k-1}} \right\}.$$

where k_n is the number of attributes left at stage n; q is selecting the number of attributes that are "interesting", and l is the smallest q such that the conditions in Equation 2.1 are still met.

5. If there is only one attribute left, stop and return it as the attribute with the highest mean, else go back to step 1.

What this procedure does basically is to take one sample at a time and check to see which attributes are still likely to be those with the highest mean. The $k_n - l$ attributes which are not likely are incrementally eliminated from consideration. Assume that there are a total of r stages before the best attribute can be picked. This final attribute has survived r stages of elimination. Let the number of attributes eliminated at stage n_i be s_i, $1 \leq i \leq s_i$, and $\sum_{i=1}^{r} s_i = k - 1$. Then since

$$\prod_{i=1}^{r} P^{*\frac{s_i}{k-1}} = P^*,$$

it is clear that the final attribute has the highest mean with probability P^* as desired. Note that any procedure $\mathcal{P}$ that eliminates attributes such that the final product of the probabilities is P^* will be a procedure consistent with the goal in Equation 2.1.

The rest of the paper proceeds to compare this approach to algorithms from Perng [74], Kao and Lai [46], Bechhofer and Goldsman [4, 5], and Edwards [30]; all of which are slight variations on the same theme. The comparison in itself is interesting, as the metric by which the algorithms are typically judged is the expected number of samples needed to pick the best attribute as the probability of a correct choice P^* approaches 1.

Most of the work in this field has lived with several assumptions that must be recognized and addressed in order to fit sequential sampling schemes to the problem of inducing models from data.

One of the major differences is that the distributions $\mathcal{N}_1, \ldots, \mathcal{N}_k$ that are being chosen from are in most cases assumed to be normal with a common known variance. In our case, we are not selecting between known, static distributions, but rather building distributions that represent the sum of our knowledge, and using these to make decisions. These distributions are unknown, and not static. The more data seen, the smaller the sample variances of the distributions are likely to become.

Second, the basic goal espoused by Equation 2.1 is flawed. The goal states that if the difference between the highest mean and the next highest mean is at least δ^*, then the answer must be at least correct with a probability of at least P^*. If the difference is less than δ^*, then a random choice is ok. This does not work in our domain, partly because the different interpretation of the distributions implies that the distributions change as more data is seen. For example, if the data so far indicates that two attributes have identical means, it does *not* necessarily mean that we can choose either one. What if very little data has been seen so far, so the distributions have large variances? Then the probability that those sample means will move as more data is seen is very large, thus spoiling the choice made at that point. What needs to happen is to incorporate the notion of *Expected Loss* into this goal, where loss will use the current means and *variances* of the distributions to predict the expected cost of an incorrect choice. This is discussed in great detail in Chapter 4.

Third, the fact that the sequential sampling strategies are based on the idea of taking one instance at a time until the condition specified in Equation 2.1 is met is disturbing. This will minimize the number of samples taken in the long run, but it will not minimize the total expected cost. In our domain, the database is already at hand, the proper goal is *not* to minimize the number of instances, but to minimize resource cost (primarily time). There will be an overhead associated with the process of extracting a sample from a database; if this overhead is amortized over a larger number of instances per sample, then we reduce the average cost of an instance. The new goal then becomes to find a sequential sampling strategy that reduces the expected cost of the computation, rather than the amount of data used.

The portion of work that is transferable is the idea of estimating the expected sample sizes required to get results of specific degrees of accuracy. Here all approaches are basically equivalent in that the general form of the equation to solved for the sample size is the same (see Section 4.2.3 for details)

$$L(S) = \int_{X_1=0}^{1} \int_{X_2=X_1}^{1} (X_2 - X_1) f_2^S(X_2) f_1^S(X_1) dX_2 dX_1.$$

The new ideas and work are introduced when finding the distributions $f_2^S(X_2)$ and $f_1^S(X_1)$, and solving the integral.

Chapter 4 addresses all of the issues stated above, producing a sampling theory that fits comfortably in the new domain of database mining.

Chapter 3

Propagating Distributions

This chapter focuses on the induction and use of the conditional probabilities in a belief network, given the graphical network structure. A straightforward approach to learning conditional probabilities of variables in a database is to infer them statistically by sampling from the database, or treating the entire database as one large sample itself [75, 23]. These values can be learned as point probabilities, intervals [33], or even as second order distributions [47, 64, 63, 94]. Distributions are generally preferable to point probabilities or intervals because they contain more information, and thus are more useful. For example, distributions give the ability to assign some degree of confidence to the inferences performed; they also allow comparison between inference solutions produced by different networks or algorithms. There is potential for much more. To date, however, there is no completely satisfactory method with the capability of propagating distributions correctly through inference; the assumptions that have been required compromise the validity or generality of the resulting learned distributions.

This chapter provides a theory based on a set of four network transformations that can produce any inference distribution in a given network, with the main assumption being that the given qualitative structure of the network is correct. The theory proves that inference distributions can be propagated without significant loss, and without debilitating assumptions.

3.1 Framework

$Pr(D, P, S)$							
		P, S					
		0,0	0,1	1,0	1,1	2,0	2,1
D	b	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
	h	θ_7	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}
	n	θ_{13}	θ_{14}	θ_{15}	θ_{16}	θ_{17}	θ_{18}

Table 3.1: **A General Joint Probability Table for the Smoker Network**
This is the joint probability table representing all possible events that could occur in the Smoker domain. Note that the θs here are actually products of the θs in the table below. For example, $\theta_1 = \theta_{19}\theta_{22}\theta_{28}$.

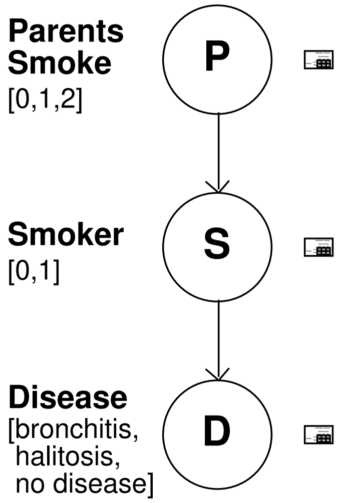


Figure 3.1: **A Smoker Belief Net**

Repeated from Chapter 1, this is a belief net showing a simplistic relation between smoking, bronchitis and having parents that smoke.

$Pr(P)$		
P	0	θ_{19}
	1	θ_{20}
	2	θ_{21}

$Pr(S\|P)$				
		P		
		0	1	2
S	0	θ_{22}	θ_{23}	θ_{24}
	1	θ_{25}	θ_{26}	θ_{27}

$Pr(D\|S)$			
		S	
		0	1
D	b	θ_{28}	θ_{29}
	h	θ_{30}	θ_{31}
	n	θ_{32}	θ_{33}

Table 3.2: **CPTs for the Smoker Network**

These are the conditional probability tables for the smoker network.

Figure 3.1 (repeated from Chapter 1) shows a simple smoker belief net that we will use to demonstrate some of the ideas described here. The CPTs for the network are repeated in Table 3.1 and Table 3.2, where the top table labeled $Pr(D, P, S)$ is actually the joint distribution over all three variables, with the following three tables being the CPTs connected to the belief network.

The problem of learning the CPTs is really the problem of learning each of the thetas shown in these tables. Each theta represents an unknown conditional probability, one per cell. The thetas within any particular column of a table are dependent: they must all add to 1. Since we learn *distributions* for each theta as opposed to point probabilities, the distributions will also be dependent.

The requirement is to be able to take the belief network and ask for any probability involving the belief network variables that comes to mind. Since a distribution for each theta is directly represented (as will become clear in the next section), it should be clear that if the value of θ_{19} ($Pr(P = 0)$) is asked for, the distribution can be returned directly. Furthermore, we would like to be able to ask for a probability not directly stored in the CPTs (and therefore not directly learned) such as $Pr(D|P)$, and still be able to return a distribution.

The next section provides a formal derivation showing what the distributions are and how they come about. Examples are given to clarify the exposition. The rest of the chapter is devoted to the question of how to answer $Pr(D|P)$ and other inferences like it.

3.1.1 A formal description

The purpose of this section is to clarify exactly what the distributions for BNI are, where they come from, and how they will be learned from samples. In particular, the beta distribution is introduced and proposed as the natural representation for the conditional probabilities that are being learned. See the Appendix for a preliminary description of the beta distribution.

We are given a database of instances D, where D might be very large. Let $S \subseteq D$ be a subsample of the database that is drawn by sampling with replacement, and let B_D be the belief net that corresponds to the underlying model from which D was drawn. There are n variables $X_1, \ldots, X_n$ represented in B_D, where variable X_i takes values from the set $x_{ik} \in \mathcal{D}_{X_i}$. A complete instance s is an element of S that assigns a value from $\mathcal{D}_{X_i}$ to every variable X_i. Let Π_i be the parents of variable X_i, ϕ_i be the set of unique instantiations of the parents Π_i, and $\phi_i[j]$ be the jth unique instantiation. Finally, $\iota_{i,j,k}$ is the combination of the parent's instantiation $\phi_i[j]$ with the variable instantiation x_{ik}.

The formal update process, the mechanism by which the conditional probability tables is learned, is derived from the following arguments. In the belief net B_D there are a set of unknown conditional probabilities $\theta_{i,j,k}$ that are being estimated for each possible instantiation $\iota_{i,j,k}$ in the network. The estimation problem can be considered one of statistical inference in which observations have been taken from a p.d.f. $f(s_l|\theta_{i,j,k})$, where $\theta_{i,j,k}$ is unknown. Take p independent random samples $s_1, \ldots, s_p$ from a distribution $f(s_l|\theta_{i,j,k})$. Let the joint p.d.f. of the p samples be

$$\begin{aligned} f_p(\mathbf{s}|\theta_{i,j,k}) &= f_p(s_1, \ldots, s_p|\theta_{i,j,k}) \\ &= f(s_1|\theta_{i,j,k}) \cdots f(s_p|\theta_{i,j,k}). \end{aligned}$$

Choose some prior distribution $\xi(\theta_{i,j,k})$ for $\theta_{i,j,k}$. The posterior distribution $\xi(\theta_{i,j,k}|\mathbf{s})$, which is the estimate of $\theta_{i,j,k}$, is then found as

$$\xi(\theta_{i,j,k}|\mathbf{s}) = \frac{f_p(\mathbf{s}|\theta_{i,j,k})\xi(\theta_{i,j,k})}{\int_\Omega f_p(\mathbf{s}|\theta_{i,j,k})\xi(\theta_{i,j,k})d\theta_{i,j,k}} \quad \text{for } \theta_{i,j,k} \in \Omega,$$

which is proportional to $f_p(\mathbf{s}|\theta_{i,j,k})\xi(\theta_{i,j,k})$.

When sampling with replacement from the database D, a natural description of the sample distribution $f(s_l|\theta_{i,j,k})$ is as a Bernoulli distribution[1]; in a relevant sample, there is a $\theta_{i,j,k}$ chance that the sample will have X_i assigned to x_{ik} (given that the parents Π_i have the assignment $\phi_i[j]$), and a $1-\theta_{i,j,k}$ chance that X_i will have a different value. With priors distributed as beta distributions, the posterior distributions will be betas as well. More precisely, if the prior is a beta with parameters a and b ($\beta(a,b)$), and we take p samples, y of which are successful (meaning $X_i = x_{ik}, \Pi_i = \phi_i[j]$), then the posterior is $\beta(a+y, b+p-y)$. An extended proof of this can be found in [26].

We store the distribution for each conditional probability by storing the sufficient statistics α and ω of a beta distribution. Though the prior in each cell of the table for variable X_i is arbitrarily set to $\beta(\alpha = 1, \omega = |\mathcal{D}_{X_i}| - 1)$, the theorems in the following section depend only on the prior being a beta. With the priors established, the induction of the quantitative structure of the network is simply a matter of incrementing the α and ω statistics of each conditional probability for each relevant sample seen. A sample is relevant to the conditional probability $Pr(X_i = x_{ik}|\Pi_i = \phi_i[j])$ if the sample is consistent with $\Pi_i = \phi_i[j]$. The α statistic is incremented if both $X_i = x_{ik}$ and $\Pi_i = \phi_i[j]$ hold in the sample; the ω statistic is incremented if $X_i \neq x_{ik}$, but $\Pi_i = \phi_i[j]$.

3.1.2 An example

Using the results is much simpler than describing them. Table 3.3 depicts the situation where the original CPTs with prior are described on the left, and the CPTs on the right show what happens after updating for one example of Parents smoke = 0, Smoker = 1 and Disease = bronchitis. Each bullet shows where the sample "hits" in each particular table.

Consider the task of using this table to find the probability that a patient has bronchitis given that his parents smoke. For this equation only, let D, S and P represent the variables Disease, Smoker and Parents smoke, and let b represent bronchitis.

$$\begin{aligned} Pr(D=b|P=0) &= \sum_{k\in\{0,1\}} Pr(D=b|S=k)Pr(S=k|P=0) \\ &= Pr(D=b|S=0)Pr(S=0|P=0) + \\ &\quad\; Pr(D=b|S=1)Pr(S=1|P=0) \end{aligned}$$

Then using the means of the beta distributions from the CPTs above as estimates of these probabilities,

$$\begin{aligned} &= (1/3)(1/3) + (2/4)(2/3) \\ &= .444 \text{ vs prior estimate of } .333 \end{aligned}$$

The cell distributions were converted into point probabilities in order to complete the computation of this inference. If instead the goal was to find the distribution of the random variable $Pr(D=b|P=0)$, we would need to be able to find a closed form for $\beta(1,2)\beta(1,2) + \beta(2,2)\beta(2,1)$. This is not easy, since in the general case the beta distribution is not conjugate across addition,

[1] A sample distribution can be equivalently described as a multinomial over the joint space, in which case the priors and posteriors would be Dirichlet distributions.

$Pr(P)$				
P	0	$\beta(1,2)\bullet$		
	1	$\beta(1,2)$		
	2	$\beta(1,2)$		
$Pr(S\|P)$				
		P		
		0	1	2
S	0	$\beta(1,1)$	$\beta(1,1)$	$\beta(1,1)$
	1	$\beta(1,1)\bullet$	$\beta(1,1)$	$\beta(1,1)$
$Pr(D\|S)$				
		S		
		0	1	
D	b	$\beta(1,2)$	$\beta(1,2)\bullet$	
	h	$\beta(1,2)$	$\beta(1,2)$	
	n	$\beta(1,2)$	$\beta(1,2)$	

$Pr(P)$				
P	0	$\beta(2,2)\bullet$		
	1	$\beta(1,3)$		
	2	$\beta(1,3)$		
$Pr(S\|P)$				
		P		
		0	1	2
S	0	$\beta(1,2)$	$\beta(1,1)$	$\beta(1,1)$
	1	$\beta(2,1)\bullet$	$\beta(1,1)$	$\beta(1,1)$
$Pr(D\|S)$				
		S		
		0	1	
D	b	$\beta(1,2)$	$\beta(2,2)\bullet$	
	h	$\beta(1,2)$	$\beta(1,3)$	
	n	$\beta(1,2)$	$\beta(1,3)$	

Table 3.3: **Updating the CPTs**

This shows the process of updating from the original CPTs on the left to the new CPTs on the right, after seeing a sample of P = 0, S = 1, D = b.

which means there is no simple closed form for the addition of two betas. Therefore, the typical approach for systems without the ability manipulate distributions is to represent the random variables representing the cell probabilities with point probabilities from the start.

Note that the original answer from the prior is .333, and after the first update the solution changes to .444. The problem is that .444 gives no information on how much the estimate can be trusted; for example, there is no information on the expected variance of this result. In fact, this number is based on the evidence of one instance in the database so one would not expect to be able to put any faith at all in this answer. If a distribution were returned here instead of a point probability, then there would be much more information available to aid in judging the quality of the result.

3.2 Assumptions

The estimate B_S of B_D that is being constructed from a sample S has an updatable beta distribution for every conditional probability that is to be explicitly stored in the network. For very simple inference tasks, like those that ask for probabilities already explicit in the network, the mean and variance of the corresponding beta can be returned as the result. But when a more general inference is desired, distributions across nodes must be combined. Without strong assumptions about independence, or without approximation, there exists no *readily apparent* way to combine them.

The trick to combining the distributions is not to look for a universal method of adding betas together, or to approximate the result, but instead to make specific arguments for specific

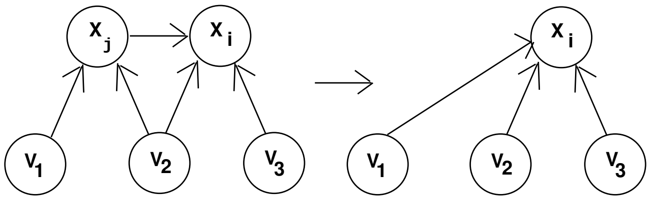


Figure 3.2: **Node Removal**

V_1 represents the set of nodes which are parents of X_j but not X_i; V_2 represents the set of parents common to both X_i and X_j; and V_3 is the set of nodes which are parents of X_i but not X_j.

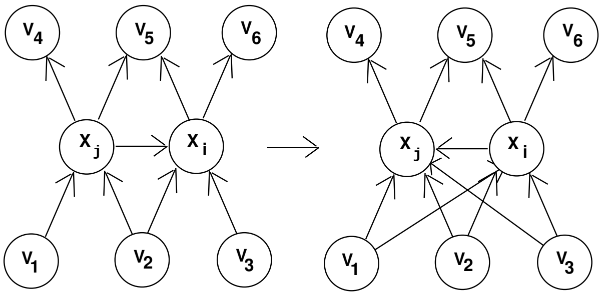


Figure 3.3: **Arc Reversal**

V_1, V_2 and V_3 are all defined as the Node Removal Figure. V_4, V_5 and V_6 are defined similarly, but represent sets of children of X_i and X_j.

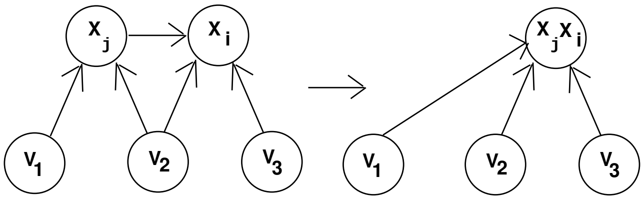


Figure 3.4: **Node Merging**

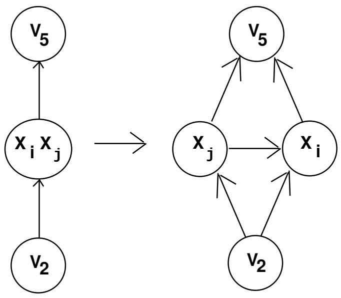


Figure 3.5: **Node Splitting**

transformations. In this way, by using the conditional independence information encoded by the transformations, and the α and ω statistics of the given distributions, we are able to compute the sufficient statistics of the new distributions. This is not a general approach; there will be a unique way to obtain each desired frequency for each particular transformation. The approach is viable, however, because we need only find the betas for a few fundamental transformations.

From Chapter 1, if Ω and Ψ are mutually exclusive subsets of the nodes in the network, then an inference problem is defined as the task of finding the conditional probability

$$Pr(\bigwedge_{X_i \in \Omega} X_i = x_{ik} \mid \bigwedge_{X_j \in \Psi} X_j = x_{jp}). \tag{3.1}$$

Olmstead and Shachter [70, 86] have shown that there are a fundamental set of transformations on an influence diagram, such that all inferences can be done in terms of these transformations. The transformations are Node Removal, Arc Reversal, Node Merging, and Node Splitting, which are shown in part in Figures 3.2, 3.3, 3.4 and 3.5. Olmstead uses these transformations to show how to propagate point probabilities through the network while doing inference. We use the same transformations in a belief network, demonstrating how to propagate the inference *distributions*, as opposed to the point probabilities.

Two main concepts form the basis of the proofs in this section. One is that the inference distribution is beta-representable. From Section 3.1.1, it is clear that when considering one cell of a conditional probability table, $Pr(X_i = x_{ik}|\Pi_i = \phi_i[j])$, the α statistic represents all of the samples that fall into that cell (all samples consistent with $X_i = x_{ik}, \Pi_i = \phi_i[j]$), and the ω statistic represents the samples that satisfied the parent variable instantiations $\phi_i[j]$, but not the value x_{ik} for variable X_i. When the distribution for $Pr(X_i = x_{ik}|\Pi_i = \phi_i[j])$ is considered in terms of the joint space over the network, it can be seen that α and ω are actually statistics over two clusters of cells in the joint space: the cluster of cells that correspond to the case where $X_i = x_{ik}$ and $\Pi_i = \phi_i[j]$, and the cluster of cells corresponding to $X_i \neq x_{ik}$ and $\Pi_i = \phi_i[j]$. A simple extension of this idea leads to the fact that any inference problem in the form of Equation 3.1 can be structured in terms of two disjoint clusters of cells over the joint space, and thus is beta representable.

The other commonality is that the proofs all require the Network Assumption.

Network Assumption: Assume that the independence conditions stated implicitly in the given qualitative structure are correct.

For the rest of the chapter, we use X_{il} to represent the instantiation of the variable X_i to the value x_{il}, or $X_i = x_{il}$. For the network fragments in Figures 3.2, 3.3, 3.4, and 3.5, the Network Assumption allows the following simplification to take place:

$$\begin{aligned} Pr(X_{il}, X_{jk}, V_{1p}, V_{2q}, V_{3r}) &= Pr(X_{il}, X_{jk}, V_{2q}, V_{3r})Pr(V_{1p}|X_{il}, X_{jk}, V_{2q}, V_{3r}) \\ &= Pr(X_{il}, X_{jk}, V_{2q}, V_{3r})Pr(V_{1p}|X_{jk}, V_{2q}) \\ &= \frac{Pr(X_{il}, X_{jk}, V_{2q}, V_{3r})Pr(V_{1p}, X_{jk}, V_{2q})}{Pr(X_{jk}, V_{2q})}. \end{aligned}$$

The Network Assumption is allowing the simplification from $Pr(V_{1p}|X_{il}, X_{jk}, V_{2q}, V_{3r})$ to $Pr(V_{1p}|X_{jk}, V_{2q})$ to take place, since all of the fragments are such that the set of variables V_1 are independent of

X_i and V_3 when given values for X_j and V_2. What this assumption allows us to do is to compute the joint over the sets of variables X_i, X_j, V_1, V_2 and V_3 by using 3 smaller joints over the same set of variables. This result turns out to be extremely useful in proving the theorems in the following section. This assumption is quite common in the community currently, although there has been some work by Madigan [59] to avoid it. It seems that his work should fit quite smoothly into this framework, while still achieving the benefit of the result above. This has not yet been tried.

There is one more assumption that is commonly made, and indeed it seems impossible to get around. The assumption is that the database represents an *unbiased* random sample of the real world, at least the real world pertaining to the set of variables we are interested in. That is, we often take the database to be an undistorted reflection of the environment, but in fact it will frequently be heavily biased by the method of data collection. For example, medical databases often do not reflect the ailments of the poorer segment of the population that does not have the resources to look properly after their health, so it would not be completely correct to draw conclusions about the general population on the basis of this data.

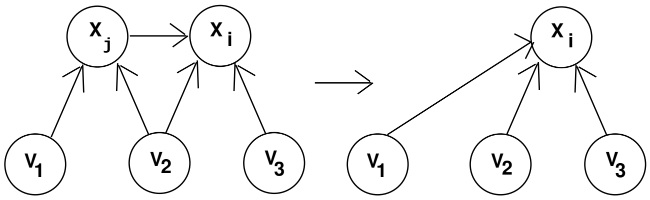
3.3 Distribution Propagation

This section gives the proofs showing the network transformations and the corresponding inference distributions. At the end of the section an outline is given of the proof that the transformations are sufficient for any inference in a belief network. In the theorems below, in order to diminish the notational nightmare, spurious subscripts have been removed except where necessary. For example, the probability $Pr(X_{il}|X_{jk}, V_{2q}, V_{3r})$ might be represented as α_1, whereas the complete description would be $\alpha_{i,l,k,q,r}$. They both refer to the alpha statistic in the cell for which $X_i = x_{il}, \ldots, V_3 = x_{3r}$. We will be using: $\#X_{il}$ to represent the *number of samples* in which variable X_i takes the value x_{il}. Also, the notation $\#\overline{X_{jk}}$ refers to the number of samples in which X_j takes any value but x_{jk}.

The Arc Reversal Theorem describes the arc reversal operation in full generality. The Node Removal, Node Splitting, and Node Merging Theorems are a bit weaker than they could be, since the network fragments given are not completely general. This simplifies the presentation of the theorems without losing the ability to do every inference in the network.

Finally, the proofs below are constructed with $\beta(0,0)$ priors, which allow the theory to be used with arbitrary priors. To add an informed prior, the theorems can be strictly applied as written; the priors will simply be treated as samples already seen. For uninformed priors, the prior must be stripped from the distribution before the transformation is performed; after the transformation the new prior can be added back to the result.

Theorem 3.3.1 Node Removal



Given:
The Network Assumption, and the network fragment shown above, where V_1 represents the set of nodes which are parents of X_j but not X_i; V_2 represents the set of parents common to both X_i and X_j; and V_3 is the set of nodes which are parents of X_i but not X_j. Also, given:

$$\begin{array}{lll} Pr(X_{il}|X_{jk}, V_{2q}, V_{3r}) & \text{distributed as} & \beta(\alpha, \omega) \\ Pr(X_{jk}|V_{1p}, V_{2q}) & \text{distributed as} & \beta(\alpha_1, \omega_1). \end{array}$$

Then,
$Pr(X_{il}|V_{1p}, V_{2q}, V_{3r})$ is distributed as $\beta(\alpha_2, \omega_2)$, where
$\alpha_2 = \sum_{d \in \mathcal{D}_{X_j}} \frac{\alpha\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}$ and $\omega_2 = \sum_{d \in \mathcal{D}_{X_j}} \frac{\omega\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}$
and the notation $\sum_{f \in \mathcal{D}_{V_1}} \alpha_1$ represents the sum of the α statistics in the table for variable X_j that correspond to cells where the variables in V_1 take any value, $X_j = x_{jk}$, and $V_2 = v_{2q}$.

Proof. We know that

$$\begin{aligned} \alpha_2 &= \#X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r}, \text{ and} \\ \omega_2 &= \#\overline{X_{il}} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r}. \end{aligned}$$

For α_2:

$$\begin{aligned} \alpha_2 &= \#X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_j}} \#X_{jd} \wedge X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r}; \end{aligned}$$

then by the network assumption,

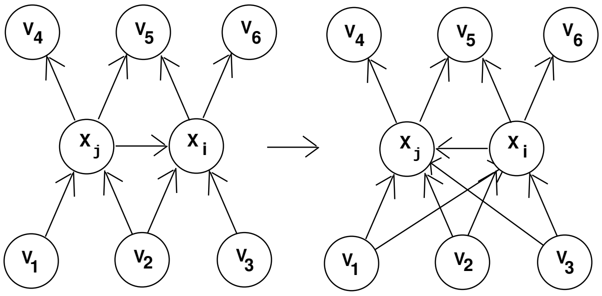
$$\begin{aligned} \alpha_2 &= \textstyle\sum_{d \in \mathcal{D}_{X_j}} \frac{(\#X_{il} \wedge X_{jd} \wedge V_{2q} \wedge V_{3r})(\#V_{1p} \wedge X_{jd} \wedge V_{2q})}{(\#X_{jd} \wedge V_{2q})} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_j}} \frac{\alpha\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}. \end{aligned}$$

For ω_2:

$$\begin{aligned} \omega_2 &= \#\overline{X_{il}} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_j}} \frac{(\#X_{jd} \wedge \overline{X_{il}} \wedge V_{2q} \wedge V_{3r})(\#X_{jd} \wedge V_{1p} \wedge V_{2q})}{(\#X_{jd} \wedge V_{2q})} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_j}} \frac{\omega\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}. \end{aligned}$$

□

Theorem 3.3.2 Arc Reversal



Given:
The Network Assumption, and the network fragment shown above, where V_1 represents the set of nodes which are parents of X_j but not X_i; V_2 represents the set of parents common to both X_i and X_j; and V_3 is the set of nodes which are parents of X_i but not X_j. V_4, V_5 and V_6 are defined similarly, as children of X_i and X_j. Also, given:

$$\begin{array}{lll} Pr(X_{il}|X_{jk}, V_{2q}, V_{3r}) & \text{distributed as} & \beta(\alpha, \omega) \\ Pr(X_{jk}|V_{1p}, V_{2q}) & \text{distributed as} & \beta(\alpha_1, \omega_1). \end{array}$$

Then,
$Pr(X_{il}|V_{1p}, V_{2q}, V_{3r})$ is distributed as $\beta(\alpha_2, \omega_2)$, where
$\alpha_2 = \sum_{d \in \mathcal{D}_{X_j}} \frac{\alpha\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}$ and $\omega_2 = \sum_{d \in \mathcal{D}_{X_j}} \frac{\omega\alpha_1}{\sum_{f \in \mathcal{D}_{V_1}} \alpha_1}$,
and
$Pr(X_{jk}|X_{il}, V_{1p}, V_{2q}, V_{3r})$ is distributed as $\beta(\alpha_3, \omega_3)$, where $\alpha_3 = \frac{\alpha\alpha_1}{\sum_{d \in \mathcal{D}_{V_1}} \alpha_1}$ and $\omega_3 = \alpha_2 - \alpha_3$.

Proof. The proof is very similar to the proof of the Node Removal Theorem. The first portion of the theorem that finds $Pr(X_{il}|V_{1p}, V_{2q}, V_{3r})$ has already been proven in the Node Removal Theorem.

For α_3:

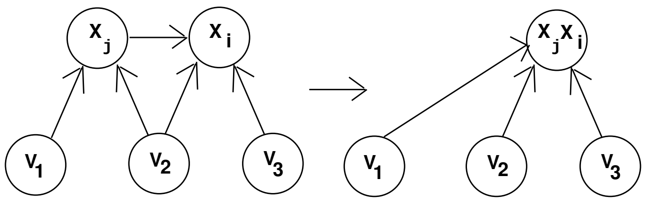
$$\begin{aligned} \alpha_3 &= \#X_{jk} \wedge X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \frac{(\#X_{il} \wedge X_{jk} \wedge V_{2q} \wedge V_{3r})(\#V_{1p} \wedge X_{jk} \wedge V_{2q})}{(\#X_{jk} \wedge V_{2q})} \\ &= \frac{\alpha\alpha_1}{\sum_{d \in \mathcal{D}_{V_1}} \alpha_1}. \end{aligned}$$

For ω_3:

$$\begin{aligned} \omega_3 &= \#\overline{X_{jk}} \wedge X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \#X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} - \#X_{jk} \wedge X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \alpha_2 - \alpha_3. \end{aligned}$$

□

Theorem 3.3.3 Node Merging



Given:
The Network Assumption, and the network fragment shown above, where V_1 represents the set of nodes which are parents of X_j but not X_i; V_2 represents the set of parents common to both X_i and X_j; and V_3 is the set of nodes which are parents of X_i but not X_j. Also, given:

$$\begin{array}{lll} Pr(X_{il}|X_{jk}, V_{2q}, V_{3,r}) & \text{distributed as} & \beta(\alpha, \omega) \\ Pr(X_{jk}|V_{1p}, V_{2q}) & \text{distributed as} & \beta(\alpha_1, \omega_1). \end{array}$$

Then,
$Pr(X_{il}, X_{jk}|V_{1p}, V_{2q}, V_{3r})$ is distributed as $\beta(\alpha_2, \omega_2)$, where $\alpha_2 = \frac{\alpha\alpha_1}{\sum_{d\in\mathcal{D}_{V_1}} \alpha_1}$ and
$\omega_2 = \frac{\alpha\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1} + \sum_{d\in\mathcal{D}_{X_j}} \frac{(\omega-\alpha)\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1}$.

Proof. The result for α_2 was proven above in the Arc Reversal theorem.
For ω_2:

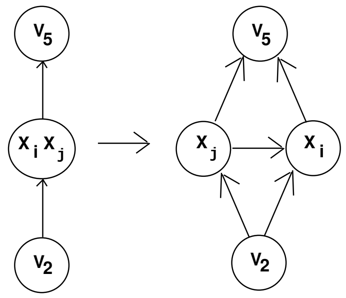
$$\begin{aligned} \omega_2 &= \#\overline{X_{il} \wedge X_{jk}} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} \\ &= \#\overline{X_{il}} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r} - \#\overline{X_{jk}} \wedge X_{il} \wedge V_{1p} \wedge V_{2q} \wedge V_{3r}, \end{aligned}$$

and from the proof of the Arc Reversal Theorem,

$$\begin{aligned} \omega_2 &= \textstyle\sum_{d\in\mathcal{D}_{X_j}} \frac{\omega\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1} - (\sum_{d\in\mathcal{D}_{X_j}} \frac{\alpha\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1} - \frac{\alpha\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1}) \\ &= \frac{\alpha\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1} + \textstyle\sum_{d\in\mathcal{D}_{X_j}} \frac{(\omega-\alpha)\alpha_1}{\sum_{f\in\mathcal{D}_{V_1}} \alpha_1}. \end{aligned}$$

□

Theorem 3.3.4 Node Splitting



Given:
The Network Assumption, and the network fragment shown above, where V_2 represents the set of parents common to both X_i and X_j; and V_5 is the set of children for both nodes. Also, given:

$$Pr(X_{il}, X_{jk}|V_{2q}) \quad \text{distributed as} \quad \beta(\alpha, \omega).$$

Then,
$Pr(X_{jk}|V_{2q})$ is distributed as $\beta(\alpha_1, \omega_1)$, where
$\alpha_1 = \sum_{d \in \mathcal{D}_{X_i}} \alpha$, and $\omega_1 = \sum_{d \in \mathcal{D}_{X_i}} \sum_{f \in \mathcal{D}_{X_j}, f \neq k} \alpha$;
and
$Pr(X_{il}|X_{jk}, V_{2q})$ is distributed as $\beta(\alpha_2, \omega_2)$, where $\alpha_2 = \alpha$, and $\omega_2 = \sum_{d \in \mathcal{D}_{X_i}, d \neq l} \alpha$.

Proof. Again the proof is very similar to the proof of the Node Removal Theorem.
For α_1:

$$\begin{aligned} \alpha_1 &= \#X_{jk} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}} \alpha. \end{aligned}$$

For ω_1:

$$\begin{aligned} \omega_1 &= \#\overline{X_{jk}} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}} \#\overline{X_{jk}} \wedge X_{id} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}} \sum_{f \in \mathcal{D}_{X_j}, f \neq k} \#X_{jf} \wedge X_{id} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}} \sum_{f \in \mathcal{D}_{X_j}, f \neq k} \alpha. \end{aligned}$$

For α_2:

$$\begin{aligned} \alpha_2 &= \#X_{il} \wedge X_{jk} \wedge V_{2q} \\ &= \alpha. \end{aligned}$$

For ω_2:

$$\begin{aligned} \omega_2 &= \#\overline{X_{il}} \wedge X_{jk} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}, d \neq l} \#X_{id} \wedge X_{jk} \wedge V_{2q} \\ &= \textstyle\sum_{d \in \mathcal{D}_{X_i}, d \neq l} \alpha. \end{aligned}$$

□

The final thing to show in this section is an outline of the proof that these transformations are sufficient for any inference in a belief network. Consider an inference in the form of Equation 3.1. Choose any source node (a node with no children) in the network, and label that the target. Choose a parent; if the parent is a part of the desired probability, use the Node Merging transformation to merge the parent with the target, otherwise use Node Removal to remove it. Do this until the only node left in the network is the target node. Then for every node in Ψ, perform the Node Splitting transformation to split off those nodes from the target. The end result is that the conditional probability table for the target will explicitly contain the inference distribution asked for in Equation 3.1. This approach would, of course, be horribly slow in practice.

3.3.1 Examples

In this section we work through several examples to get a better feeling for what these theorems mean, and how to apply them.

	fo, bp	$fo, \mathbf{bp}$	$\mathbf{fo}, bp$	$\mathbf{fo}, \mathbf{bp}$
lo, do, hb	4	10	0	1
$lo, do, \mathbf{hb}$	0	1	0	0
$lo, \mathbf{do}, hb$	0	1	0	0
$lo, \mathbf{do}, \mathbf{hb}$	0	7	0	5
$\mathbf{lo}, do, hb$	1	4	3	10
$\mathbf{lo}, do, \mathbf{hb}$	0	1	0	1
$\mathbf{lo}, \mathbf{do}, hb$	0	0	0	1
$\mathbf{lo}, \mathbf{do}, \mathbf{hb}$	0	3	3	44

Table 3.4: **The Frequency Distribution for FO,BP,LO,DO,HB**
This shows the total number of events that occurred for every possible variable instantiation.

Dog-Out Example

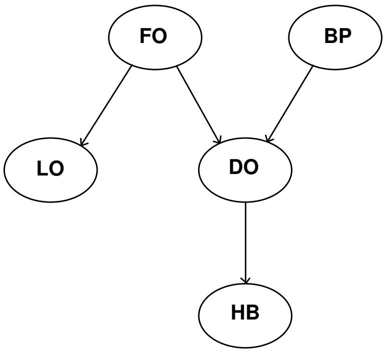


Figure 3.6: **The Dog-Out Example**
Taken from Charniak, all variables are binary. FO = fo when the family is out, $\mathbf{fo}$ when not. Similarly, BP = bp when the dog has bowel problems, LO = lo when the outside lights are on, DO = do when the dog is out, and HB = hb when you can hear the dog barking.

The network for the first example is shown in Figure 3.6 taken from Charniak [15]. It is a qualitative model used by a fictitious Mr. Smith to predict whether his family is out or not. The prediction is based on whether the outdoor lights are on, and whether the dog's barking can be heard. When the family leaves, they often put the dog out, and turn the outdoor light on. Also, when the dog is having bowel problems, he might be put out. An early indication of the dog being out is whether or not his barking can be heard.

Instead of attempting to construct the quantitative structure from memory of past experiences, Mr. Smith decides to start with an uninformative prior, and record his experiences daily (one can consider this an expensive sampling process for the more general task of sampling from a large scientific or business database). After 3 months, the sum total of his experience is represented by the frequency of each occurrence in the joint space, as shown in Table 3.4.

$Pr(\mathrm{FO})$	
fo	$\beta(33,69)$
fo	$\beta(69,33)$

$Pr(\mathrm{BP})$	
bp	$\beta(12,90)$
bp	$\beta(90,12)$

$Pr(\mathrm{HB}\vert\mathrm{DO})$		
	do	**do**
hb	$\beta(34,4)$	$\beta(3,63)$
hb	$\beta(4,34)$	$\beta(63,3)$

$Pr(\mathrm{LO}\vert\mathrm{FO})$		
	fo	**fo**
lo	$\beta(24,10)$	$\beta(7,63)$
lo	$\beta(10,24)$	$\beta(63,7)$

$Pr(\mathrm{DO}\vert\mathrm{FO,BP})$				
	fo, bp	fo, **bp**	**fo**,bp	**fo**, **bp**
do	$\beta(6,1)$	$\beta(17,12)$	$\beta(4,4)$	$\beta(13,51)$
do	$\beta(1,6)$	$\beta(12,17)$	$\beta(4,4)$	$\beta(51,13)$

Table 3.5: **Conditional Probability Tables Dog-Out Example**
These tables are what would actually be stored, given the data as seen in Table 3.4.

In general, tables representing the joint probability or frequency space will not be available, because they are usually far too large to represent outright. That is the raison d'être for belief networks: to efficiently represent the information in the joint space with a set of conditional probability tables, based on the independence assumptions characterized by the graphical layout (the Network Assumption). What is shown in Table 3.5 is the set of conditional probability tables that would be generated by BNI for the data. For example, the entry for $Pr(hb|do) = \beta(34,4)$ comes from the prior distribution of $\beta(1,1)$ in each cell, plus 33 instances in which the dog was barking while outside, and 3 instances in which the dog was outside but not barking.

For this example, we compute the $Pr(fo|lo, hb)$, and watch how the inference distribution is propagated throughout the transformation process. One possible set of transformations that can be performed on Figure 3.6 to get the desired distribution is shown in Figure 3.7. This transformation process proceeds as follows:

1. Arc Reversal: FO $\rightarrow$ LO. Following Theorem 3.3.2, the resulting distribution for $Pr(\mathrm{FO}{=}\,fo|$ $\mathrm{LO}{=}lo) = \beta(24,7)$. There are five other distributions (the rest of the table for $Pr(\mathrm{FO}|\mathrm{LO})$, and the table for $Pr(\mathrm{LO})$) that could be specified at this point on the basis of Theorem 3.3.2, but this is the only one of interest for this example.

2. Node Removal: BP. The distribution of $Pr(\mathrm{DO}{=}do|\ \mathrm{FO}{=}fo) = \beta(22,12)$.

3. Arc Reversal: DO $\rightarrow$ HB. The distribution of $Pr(\mathrm{DO}{=}do|\mathrm{FO}{=}fo,\mathrm{HB}{=}hb) = \beta(20.25, 1.34)$. This is the first time the Network Assumption has cost accuracy. If we look at Table 3.4, we can see that the actual distribution based on the samples seen is $\beta(20,2)$. The joint table, however, is not stored, so we must rely on the conditional probability tables for partial information, and the Network Assumption for the rest. In this case, an independence assumption

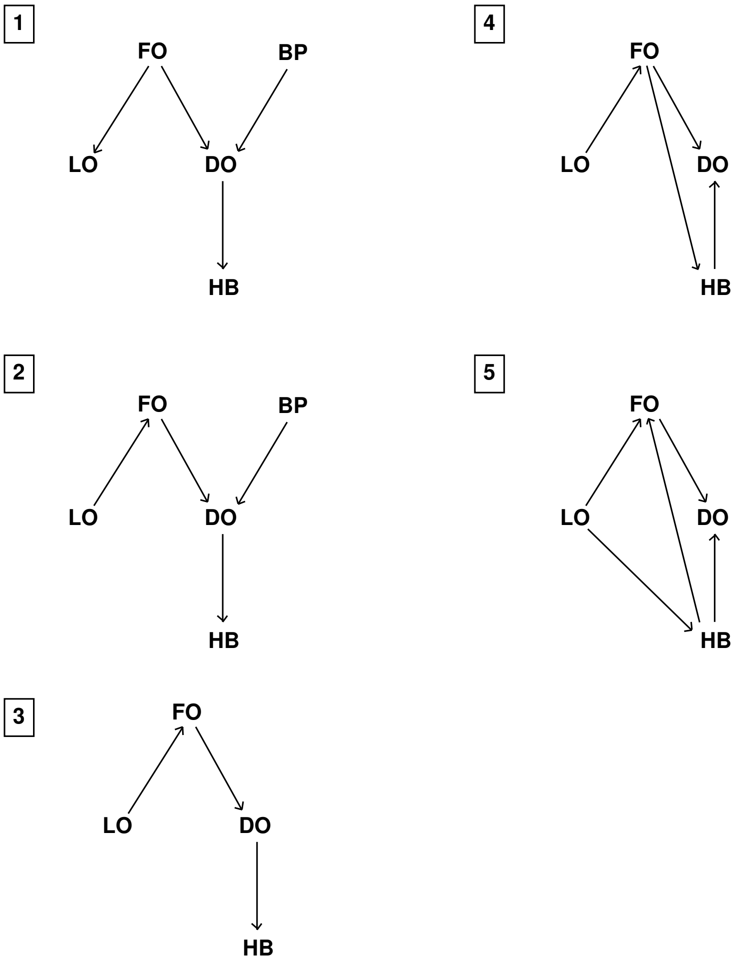


Figure 3.7: **Transformation Steps to Find the** $Pr(fo|lo, hb)$
The sequence of steps is labeled in each upper left hand corner.

was made that was not 100% supported by the data, and so the discrepancy. This is not too surprising, since it would be rare for a random sample of any size to completely support the independence conditions. Larger samples would, however, tend to provide stronger support.

4. Arc Reversal: FO $\rightarrow$ HB. The distribution of $Pr(\text{FO}{=}fo|\ \text{LO}{=}lo,\text{HB}{=}hb) = \beta(15.08, 2.36)$. This is the final result of the computation. Comparing this to the distribution we would get if Table 3.4 were available, we can see that the results are reasonable ($\beta(16, 2)$).

It is widely accepted that as the inference chain grows, the error in the distributions being maintained will also grow. This statement is a bit too strong. As pointed out by Klieter [47], there are two competing forces that will determine whether error or expertise is growing:

- As the inference chain grows, the error at each step is compounded, thus providing a force for increasing error.

- As the chain grows, certain types of inference (for example, marginalization) implicitly add samples from other parts of the network to the information currently at hand. Since the variance of the beta distribution is

$$\begin{aligned} var(\beta(\alpha,\omega)) &= \frac{\alpha\omega}{(\alpha+\omega)^2(\alpha+\omega+1)} \\ &< \frac{(\alpha+\omega)^2}{(\alpha+\omega)^2(\alpha+\omega+1)} \\ &= \frac{1}{\alpha+\omega+1,} \end{aligned}$$

  the variance will tend to decrease linearly as samples are added to the system, thus providing a force for increasing expertise.

The magnitude of the first force depends on how the distributions are propagated. The larger the error in each step, the stronger this force. For the theory presented here, the source of the error is in the violation of the Network Assumption, in particular due to the sampled data not conforming to the independence claims made by the qualitative structure. As the size of the sampled data grows, this source of error will drop off fairly rapidly. It seems more reasonable to say that there exists a probabilistic *boundary* on the sample size [65]. Below this boundary the error will grow as the inference chain grows, above it, expertise will grow instead.

Car Insurance Example

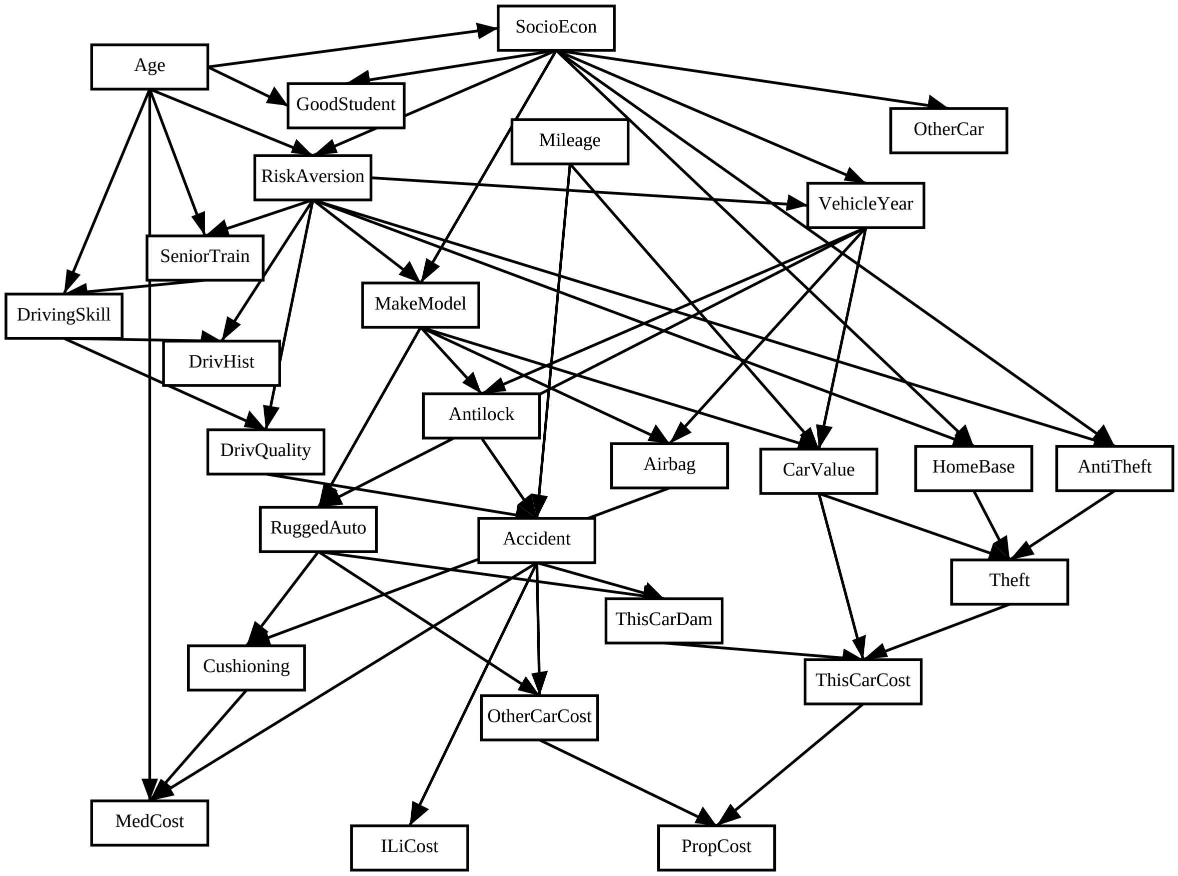


Figure 3.8: **Car Insurance Database**

This is a model of an insurance database, it includes binary, nominal, discrete and continuous variables.

An interesting question is whether or not the operations are order independent. In the car insurance belief net shown in Figure 3.8, suppose the goal is to find the probability Pr(DrivingSkill = expert | GoodStudent = false). There are many different ways to apply the four transformations in order to come up with a network which can answer this question; it would be good if all the paths produced the same distributions as outputs. We in fact claim that the transformations are order independent. We back this up with an example based on an implementation of the transformation theorems, and a simple argument that explains why this is as it should be.

Similar to the data in the dog-out example above, the CPTs in the car insurance network were learned with the updating method discussed in Section 3.1.1, this time using 500 samples. Again, as before, by looking at the actual data the "true" beta distribution given the data can be determined for the probability at hand by counting the actual number of samples for which DrivingSkill = expert and GoodStudent = false; and DrivingSkill != expert and GoodStudent = false. We end up with a target of $\beta(54, 426)$.

The Arc Reversal Theorem alone was applied to generate the two possible solutions shown

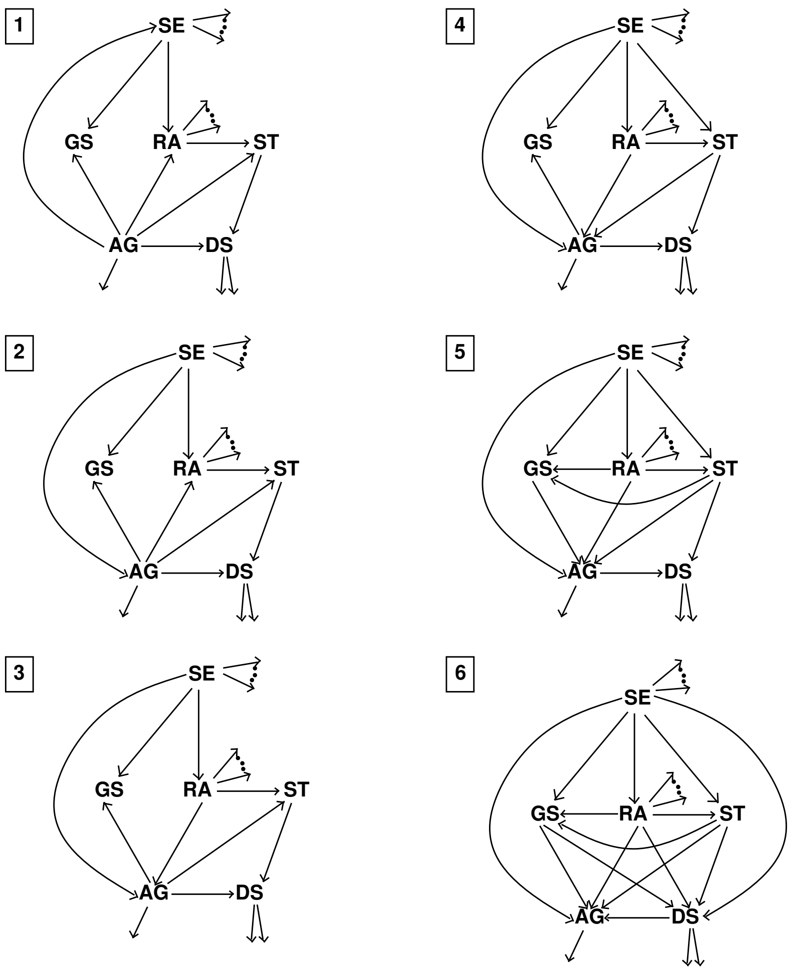


Figure 3.9: **Transformation #1 for Pr(DrivingSkill=expert | GoodStudent=false)**
This shows the application of the Arc Reversal Theorem to the sequence of arcs Age → SocioEcon, Age → RiskAversion, Age → SeniorTrain, Age → GoodStudent and Age → DrivingSkill. Transformations at steps 4 and 5 establish GoodStudent as a parent of DrivingSkill, the rest are to make sure that no cycles get created.

in Figures 3.9 and 3.10. These figures are such that SE represents the node SocioEcon, AG Age, RA RiskAversion, DS DrivingSkill, ST SeniorTraining and GS GoodStudent. The rest of the network has not been shown as it is not directly affected by the transformations that were applied. The goal is to apply a sequence of transformations that results in having GS as a parent of DS. This condition insures that the CPT for DS would then have all of the information necessary to determine the distribution for Pr(DS=expert | GS=false). The Arc Reversal Theorem works in this case because it adds links such that the parents of the nodes that are being reversed are all shared. For example in Figure 3.10 at step 4, reversing the link SE → DS adds the link GS → DS since GS is a parent of SE.

The end result of applying these transformations is that for both Figure 3.9 and Figure 3.10 we get the distribution $\beta(57.81, 425.19)$ (or without a prior, $\beta(56.81,\ 423.19)$). In this case, the

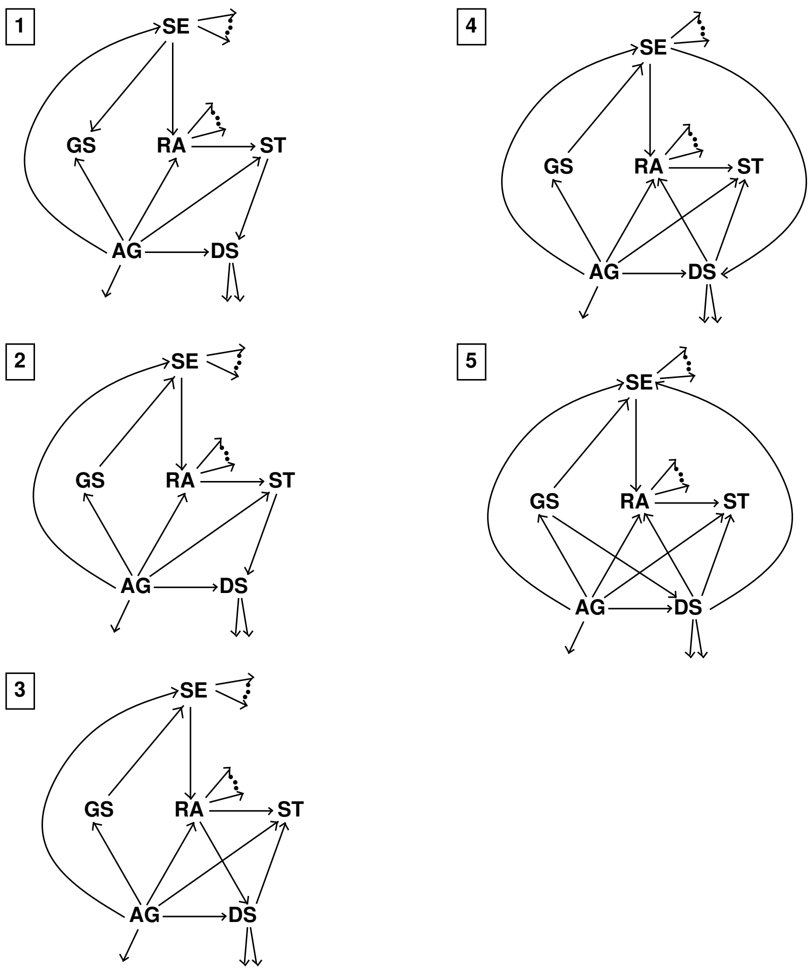


Figure 3.10: **Transformation #2 for Pr(DrivingSkill=expert | GoodStudent=false)**
This reverses the sequence of arcs SocioEcon → GoodStudent, SeniorTrain → DrivingSkill, RiskAversion → DrivingSkill, and SocioEcon → DrivingSkill.

resulting distribution is the same for either path. The same tables can be used to provide, for example, the distribution on an expert driver given that driver is a good student student; in this case the transformations returned $\beta(2.19, 20.81)$ (or $\beta(1.19, 18.81)$ without a prior), whereas the data tells us it should be $\beta(4, 16)$.

The two transformations in this example reverse completely different sets of arcs, and end up making modifications to the CPTs that are significantly different. For example, the CPT for DrivingSkill under the first set of transformations has 192 entries in it, whereas under the second set of transformations it has only 18. Yet, the end result is to provide the same distribution on the question at hand.

The reason that different sets of transformations produce the same distributions is really very simple. The transformations themselves are not inflicting any estimations on the system, they are mathematically precise functions in the domain for which they are designed, in much the same way that multiplication and division are. The estimates and inaccuracies are from two sources, the Network Assumption, and the fact that the data seen at any point is a random quantity and thus can not be expected to conform exactly to the independencies inherent in the given structure (even if the structure is correct). When asking for the distribution of some probability $Pr(X|Y)$, this depends only on the structure of the relationship between the variables X and Y, thus any inaccuracies will stem from the improper modeling of that structure, or from the random nature of the random sample. If the different processes used to derive the distribution accurately reflect the structure given, then no matter which method or process is used, the results will be the same. This is where other approaches to this problem fall short, as their error is dependent not only on model error and random sample error, but also on process error.

One more point of interest is to notice that the distribution for the bad student is much more accurate than that for the good student. The reason for this, as discussed in the previous example, is that very few samples have been seen for the good student case (20, compared to 480 for the bad student) and so the data is not yet accurately reflecting the independency structure assumed by the network.

3.4 Discussion

In systems that learn belief networks by induction, there is a need to produce inference distributions rather than simple point probabilities in order to give some feeling for the degree confidence that can be placed in the result. It is impossible, however, to produce distributions that accurately reflect all of the data seen without retaining the equivalent of the joint probability space for the network; there are too many unknowns, too few equations. It is necessary, then, to make assumptions or approximations that allow us to propagate the distributions through inference. These assumptions have been strong in the past, compromising the integrity of the resulting distributions.

What this chapter shows is that by using only the Network Assumption, the inference distributions can be propagated for any inference possible in a belief network, by specifying how to update the sufficient statistics on the beta distributions for four fundamental network transformations. This chapter has also argued that under some types of inference, error will tend to decrease as the inference chain grows, as long as the belief network has been constructed from a sufficient number of instances.

For future work, the manipulations must be written in terms of realistic "fast" inference techniques like those explored in Lauritzen and Spiegelhalter [52, 51]. Furthermore, learning the quantitative structure of a belief net does not end with the management of the beta distributions; we need to examine the possibility of learning good conditional probability tables with few samples. The quality of inference done in an induced network is dependent on the level to which the joint probability space has been explored. The fact of the matter is that for many domains, resources (the size of the database, the time available) will not be sufficient to adequately explore the joint probability space for a belief network; the space is too large, and even in a small space there are bound to be low-probability columns in CPTs that rarely see the light of data. The job is made just that much harder when the assumption is made that every column in the CPTs is independent, as is done in the method explored above. In Chapter 5 we examine in depth other methods of learning this space that avoid the assumption of independence.

Chapter 4

Decision Support

This chapter describes the theory behind predicting optimal sample sizes required to construct models of the data that satisfy user requirements. The first section describes the motivation behind this type of work. The second section presents the mathematics for producing maximum utility sample sizes in the domain of decision tree induction. The same concepts are then applied to the construction of belief networks.

4.1 Motivation

Real world databases are often established in order to answer a set of particular questions about the environment described by the data, or to aid in making a choice between a set of possible actions. Unfortunately, several problems of scale arise when using the standard algorithms on the huge training sets offered by very large corporate and scientific databases. Extracting, handling, and learning from a large training set can represent a very considerable effort (sometimes days or more of both human and machine time). In some applications, the benefit of increased accuracy obtained from the additional data justifies this expense, but often the economically rational decision is to use only a subset of the available training instances. For example, if the information needed is not extremely critical, a user might be just as happy with an answer of $p \pm .1$ as with $p \pm .01$, thus she can take a smaller sample and get much quicker results. What sample size is expected to be necessary to produce inferences that are within a specified expected error? Furthermore, what sequence of subsamples will be the most economical way to proceed in order to produce this sample?

As another example, in some critical areas one might find several different models being constructed for the same domain, in hopes of reducing the chance of unsuitable or incorrect results. If the models end up suggesting different courses of action, how should one determine which action to take? Or more to the point of this dissertation, how accurate must the CPTs be in order to distinguish the set of inferences to within a specifiable degree of confidence? How accurate must they be to make the results practically indistinguishable, so that either action is as good as the other? Ultimately, we wish to control the tradeoff between the cost of induction (space and time) from larger training sets and the increased accuracy they provide. The current state of the art in AI lacks a sound and principled way to do this. This chapter makes significant progress towards that goal, providing both an analytical framework within which to consider these questions and an

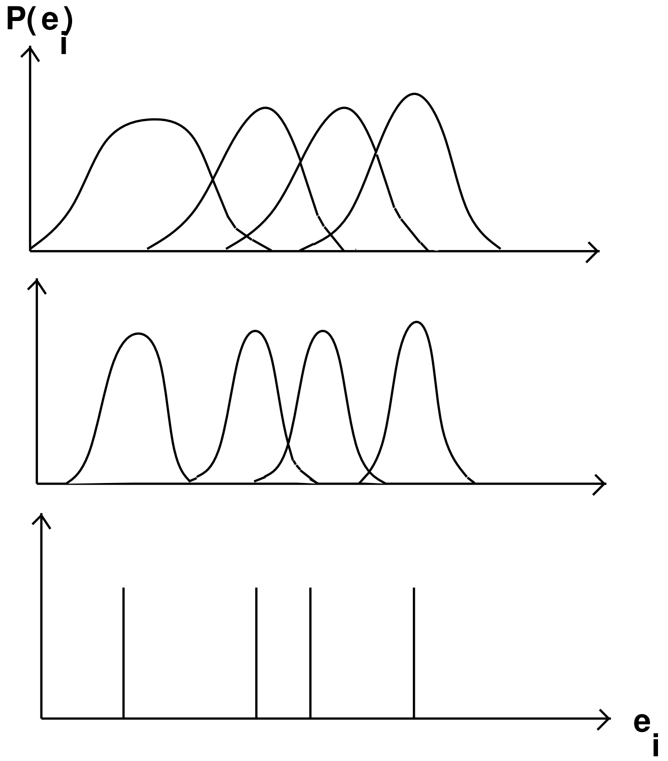


Figure 4.1: **Action Success Rate Distributions**
Probability distributions for the chance of success of 4 actions for small, medium and complete samples.

application of the concepts that is successful in empirical tests.

Figure 4.1 illustrates these concepts for four possible actions. The top figure shows a typical situation with a small sample, where the distributions plotted represent the total knowledge to date of the variables in question. The second shows the situation after more examples have been examined, and the third shows the situation after an infinite number of samples. When trying to pick the best action, it is clear that a choice made using the small sample is quite likely to be wrong, whereas we probably don't need to wait until the database is exhausted to be reasonably sure of the right choice. This shows a strong analogy to the problem of picking the best move in game-playing, when only the estimates of move values are available to work with. The techniques used here are thus similar in concept to those in [82, 40].

4.2 Application to Decision Trees

This section concentrates on building the theory to control decision tree induction. The main theme of this dissertation is the induction of models from data. Although most of the results apply primarily to belief networks, we put these networks aside for this section and study the induction of decision trees. This problem is very interesting in its own right, as decision trees are one of the most successful induction techniques in industry. Section 4.2.5 contains some rather extensive empirical results that demonstrate the effectiveness of the approach. In Section 4.3 we show that the results obtained for decision trees are directly applicable to belief networks.

The most immediate application of controlling the sample sizes is to speed up the basic model induction algorithm. For example, in the domain of decision tree induction, an earlier approach to this problem [13] shows that the algorithm can be reduced from roughly quadratic to linear time (in the number of training examples) without significant loss of accuracy by using sequential samples called peepholes. Two ad-hoc components of that method can now be placed on a sound analytical basis: the choice of the sizes for the peepholes, and the determination of whether an attribute should be discarded from consideration on larger peepholes. This second point brings out another interesting result of being able to characterize the expected gains of samples of a given size; the idea can be applied to the task of deciding whether an attribute should be discarded completely from the induction process by monitoring how it compares to alternatives in situations of interest. This allows comparatively ineffective or redundant attributes to be weeded out, thus reducing learning time as training set sizes are scaled up. This can be done automatically, or interactively with a human expert.

The following sections show how to calculate both the distributions and the expected information loss associated with selecting an attribute on the basis of a subsample. Given this information, we can calculate the expected increase in the quality of the selected attribute from further sampling, compare it to the cost of sampling, and thus derive an optimal stopping point.

4.2.1 Overview

A top down decision tree learning algorithm [78] builds a decision tree classifier from a data set $S = \{s_1 \ldots s_s\}$ using attributes $\mathcal{A} = \{A_1 \ldots A_m\}$. It operates by first selecting an attribute to form the root of the tree, then splitting the data set according to the attribute value and applying itself recursively to each subset thus formed. For large training sets, the major costs arise in selecting an attribute for the root of the current subtree, and in choosing appropriate threshold values for numeric attributes.

Attributes are normally selected using information gain calculations. The information gain of an attribute A_i given a sample S of size $|S|$ is

$$g(A_i) = I(p, n) - e_i$$

where p, n are the numbers of positive and negative examples in the sample; $I(p, n)$ is the total information (in bits, attribute independent) required to classify a random example, given by

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n};$$

and e_i is the expected amount of information needed to classify a random example (the information content) given information about the value of A_i. Note that over the set of attributes, $I(p, n)$ is constant; thus for a relative measure of the information gain of the attributes, we need only consider e_i. If we assume, for the sake of simplicity of exposition, that the attributes are binary with values $+$ and $-$, then (with
$|S| = p_i^+ + n_i^+ + p_i^- + n_i^-$)

$$e_i = \frac{p_i^+ + n_i^+}{S} I(p_i^+, n_i^+) + \frac{p_i^- + n_i^-}{S} I(p_i^-, n_i^-)$$

$$
\begin{aligned}
&= -\frac{p_i^+}{S}\log_2(\frac{p_i^+}{p_i^+ + n_i^+}) - \frac{n_i^+}{S}\log_2(\frac{n_i^+}{p_i^+ + n_i^+}) - \\
&\quad \frac{p_i^-}{S}\log_2(\frac{p_i^-}{p_i^- + n_i^-}) - \frac{n_i^-}{S}\log_2(\frac{n_i^-}{p_i^- + n_i^-}). \qquad (4.1)
\end{aligned}
$$

If examples are vectors that include a classification p or n, and +/- values for each attribute, then p_i^+ is a example classified as p (positive) where attribute A_i has the value +.

The method works by considering subsamples of the entire data set. Given a subsample, we estimate the actual information content of each attribute with Equation 4.1. From this, we can derive the conditional probability distribution for the true information content given the estimated version. It turns out that the conditional probability distribution is in fact approximately normal.

4.2.2 Information Content Distributions

Let $X_i^{\mathcal{T}}$ be the true information content of A_i if an infinite sample were given. This is unknown, so let X_i be the random variable that estimates this quantity based on some subsample S, and let $f_i^S(X_i)$ be the distribution of X_i on that sample. Finally, let $\hat{e_i}$ be the information content *given* a sample S as per Equation 4.1. Like many sample statistics, X_i is approximately normally distributed around the information content e_i on an infinite sample, which we estimate with $\hat{e_i}$ calculated from the subsample:[1]

$$f_i^S(X_i) \approx N[\hat{e_i}, \sigma_i]. \qquad (4.2)$$

The variance σ_i for the distribution on the sample can be determined exactly. Let R_1 be the random variable that represents the true fraction of positive instances that attribute A_i classifies as + (so $\frac{p_i^+}{S}$ is the current best estimate of R_1). Also, let R_2, R_3 and R_4 be defined similarly, with estimates $\frac{n_i^+}{S}$, $\frac{p_i^-}{S}$, and $\frac{n_i^-}{S}$. Then, from Equation 4.1, the variance is given as

$$
\begin{aligned}
\sigma_i^2 &= var(R_1\log_2(\frac{R_1}{R_1+R_2}) + R_2\log_2(\frac{R_2}{R_1+R_2}) + \qquad (4.3)\\
&\quad R_3\log_2(\frac{R_3}{R_3+R_4}) + R_4\log_2(\frac{R_4}{R_3+R_4})).
\end{aligned}
$$

Note that for any attribute A_i, each example in the sample has 4 possible outcomes, p_i^+, n_i^+, p_i^- and n_i^-. This is analogous to sampling with replacement from an urn filled with balls of 4 different colors. In this case the natural distribution to describe the process is a Dirichlet (see the Appendix for more information on the Dirichlet). Therefore, the joint distribution $f(R_1, R_2, R_3, R_4)$ is a Dirichlet, and the marginal distribution of each variable is a beta (for example, R_1 is distributed as a beta with the parameters p_i^+ and $S - p_i^+$). We can solve for σ_i^2 exactly with the use of the digamma function (Ψ) and its derivative. Keep in mind that

$$var(Z) = E(Z^2) - E(Z)^2.$$

[1]Empirical tests on the goodness of fit of the normal to the actual distribution show that though not exact, the fit is reasonably good.

Now, break up the variance expression in Equation 4.3 into six terms: $Z_1 = R_1 log_2(R_1)$, $Z_2 = R_2 log_2(R_2), \ldots, Z_6 = R_6 log_2(R_6)$, where R_5 is the random var representing $(p_i^+ + n_i^+)/S$, and R_6 is $(p_i^- + n_i^-)/S$. The expression for the variance of z given above is more descriptively written as $var(z) = E((Z_1 + Z_2 + Z_3 + Z_4 - Z_5 - Z_6)^2) - E(Z_1 + Z_2 + Z_3 + Z_4 - Z_5 - Z_6)^2$. This has about 30 individual terms in it, terms that take the forms: $E(R_1^2 log_2(R_1)^2)$, $E(R_1 R_2 log_2(R_1) log_2(R_2))$, and $E(R_1 R_5 log_2(R_1) log_2(R_5))$.

The trick is to find exact expressions for $E(\log_2 R_1)$ and $E(R_1 \log_2 R_1)$, then manipulate these expressions to find σ_i^2. We will sketch the derivation of these quantities.

First, we know that $\log_2 R_1 = ln(R_1)/log_2(e)$, and therefore by the definition of Expectation and the expanded form of the beta distribution (see the Appendix):

$$E(\log_2 R_1) = \frac{E(\ln R_1)}{\log_2 e} = \frac{\int \ln(R_1) R_1^{p_i^+ - 1}(1 - R_1)^{|S| - p_i^+ - 1} dR_1}{\beta(p_i^+, |S| - p_i^+) log_2 e}.$$

The next critical piece of information (based on the chain rule of partial differentiation) is:

$$\begin{aligned} \frac{\partial}{\partial a_1} \ln \beta(a_1, a_2) &= \frac{\partial}{\partial \beta(a_1, a_2)} \ln \beta(a_1, a_2) \frac{\partial}{\partial a_1} \beta(a_1, a_2) \\ &= \frac{\int \frac{\partial}{\partial a_1} R_1^{a_1 - 1}(1 - R_1)^{|S| - a_1 - 1} dR_1}{\beta(a_1, |S| - a_1)} \\ &= \frac{\int \ln(R_1) R_1^{a_1 - 1}(1 - R_1)^{|S| - a_1 - 1} dR_1}{\beta(a_1, |S| - a_1)}. \end{aligned}$$

The second step in the above derivation is a bit tricky. It is based on the fact that $\frac{\partial \ln(f)}{\partial f} = \frac{1}{f}$ for any function f, and that $a_1 + a_2 = |S|$. Also, the third step is based on the fact that $\frac{\partial}{\partial a} x^a = \ln(x) x^a$. It follows that

$$E(log_2 R_1) = \frac{1}{log_2 e} \frac{\partial}{\partial p_i^+} \ln \beta(p_i^+, |S| - p_i^+).$$

Another form for $\frac{\partial}{\partial a_1} \ln \beta(a_1, a_2)$ that is very useful is in terms of the digamma function,

$$\begin{aligned} \frac{\partial}{\partial a_1} \ln \beta(a_1, a_2) &= \frac{\partial}{\partial a_1} \ln \frac{\Gamma(a_1)\Gamma(a_2)}{\Gamma(a_1 + a_2)} \\ &= \Psi(a_1) - \Psi(a_1 + a_2), \end{aligned}$$

giving

$$E(log_2 R_1) = \frac{1}{log_2 e}(\Psi(p_i^+) - \Psi(S)). \tag{4.4}$$

The main advantage of this formulation of $E(log_2 R_1)$ and ultimately of the variance is that the digamma function and its derivative (which shows up for terms like $E(r^2 (log_2 r)^2)$) has several quick and accurate approximations [1], which lead to fast computation of variances that are as precise as desired.

The derivation of $E(R_1 \log_2 R_1)$ follows in a similar manner, from the two pieces of information:

$$E(R_1 \log_2 R_1) = \frac{\int R_1 \ln(R_1) R_1^{p_i^+ - 1}(1 - R_1)^{|S| - p_i^+ - 1} dR_1}{\beta(p_i^+, |S| - p_i^+) log_2 e}$$

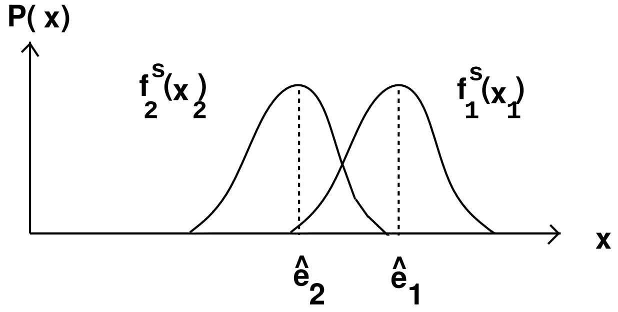


Figure 4.2: **Expected Loss**

The expected information of attribute A_1 is $\hat{e}_1$, so it appears that A_1 is the better of the two choices. But, a loss may occur if A_2 is actually better. In this graph, that would happen if the actual information of A_2 is on the right end of the distribution, while the information for A_1 is out on the left end.

$$\begin{aligned}\frac{\partial}{\partial a_1} \ln \beta(a_1+1, a_2) &= \frac{\int R_1 \ln(R_1) R_1^{a_1-1}(1-R_1)^{|S|-a_1-1} dR_1}{\beta(a_1+1, |S|-a_1)} \\ &= \Psi(a_1+1) - \Psi(a_1+a_2+1).\end{aligned}$$

The result is constructed as

$$E(R_1 \log_2 R_1) = \frac{1}{log_2 e} \frac{p_i^+}{|S|}(\Psi(p_i^+ + 1) - \Psi(|S|+1)). \tag{4.5}$$

The $\frac{p_i^+}{|S|}$ term comes from $\frac{\beta(p_i^+ + 1, |S|-p_i^+)}{\beta(p_i^+, |S|-p_i^+)}$.

Simple extensions can made to Equations 4.4 and 4.5 to complete the solution for σ_i^2. We do not show the full form of σ_i^2 since reproducing the 30-odd equations would be a bit tedious, and furthermore the information given above makes them easily reproducible. What has been shown then is how to compute the variance for the distribution of the information content of a sample as shown in Equation 4.2. The next section shows how to use this distribution to compute the expected sample sizes required to make decisions of specifiable qualities.

4.2.3 Expected Loss

Consider choosing between just two attributes A_1 and A_2, where we assume without loss of generality that the estimated gain of A_1 is currently higher than that of A_2 (Figure 4.2). What is the expected loss in information gain associated with choosing A_1?

If A_1 has true gain X_1, and A_2 has true gain X_2, then if $X_2 > X_1$, the loss is $X_2 - X_1$. Otherwise, there is no loss. Hence the expected loss is given by[2]

$$L(S) = \int_{X_1=0}^{1} \int_{X_2=X_1}^{1} (X_2 - X_1) f_2^S(X_2) f_1^S(X_1) dX_2 dX_1. \tag{4.6}$$

[2]This assumes independence of the attribute gains; that is to say that knowledge of the gain of one attribute does not affect the prediction of the gain of the second attribute.

Inserting the normal approximations for $f_1^S(X_1)$ and $f_2^S(X_2)$ obtained above, and realizing that $Y = X_2 - X_1$ is the normal $N(\hat{e} = \hat{e_2} - \hat{e_1}, \sigma = \sqrt{\sigma_1^2 + \sigma_2^2})$, we get

$$\hat{L}(S) \quad = \quad \frac{1}{\sigma\sqrt{2\pi}} \int_{y=0}^{1} y e^{-\frac{(y-\hat{e})^2}{2\sigma^2}} dy. \tag{4.7}$$

This use of Y is an approximation as well because it is possible for X_1 or X_2 as normals to be greater than 1 (which is not a valid information content), but for
$Y = X_2 - X_1 \in [0, 1]$, and thus be counted in Equation 4.7. The integral is easy to solve, giving:

$$\hat{L}(S) \quad = \quad \frac{\sigma}{\sqrt{2\pi}}(e^{-\hat{e}^2/2\sigma^2} - e^{-(1-\hat{e}^2)/2\sigma^2}) + \frac{\hat{e}}{2}(erf(\frac{1-\hat{e}}{\sqrt{2}\sigma}) + erf(\frac{\hat{e}}{\sqrt{2}\sigma})). \tag{4.8}$$

Note that this equation provides the connection between the sample size (which is embedded in σ), the width of the distributions and the expected loss in information gain that can be attributed to selecting the current best attribute. This approximation begins to degrade only when the information content comes to within a standard deviation or two of 1, at which point $\hat{L}(S)$ provides an upper bound on $L(S)$. Because $\hat{L}(S) \geq L(S)$, the solution proposed here will tend to be somewhat conservative, and thus tend to take larger samples than required. However, the empirical results in Section 4.2.5 show that this effect is barely noticeable.

4.2.4 Sampling Strategy

To find the optimal stopping point, we need information on the expected quality of the tree, the cost of sampling, and a utility measure specifying what the user is willing to pay for different quality trees. As a preliminary cut, we use the expected loss $\hat{L}(S)$ as the quality measure (the smaller the loss, the higher the quality), and allow only the simplest form of a utility measure, a single point, the *loss tolerance*. This measure is such that a decision made with a quality worse than the loss tolerance has no value, and a decision made with quality greater than or equal to the loss tolerance has full value.

Given this information, a simple approach to reach the optimal stopping point (the one-shot approach) is to calculate the expected required sample size (ERSS) needed to reach the loss tolerance, and then to take a sample of that size. This approach has drawbacks, but the calculation of ERSS is useful both in itself, and for the next section.

The first step is to find the variance σ_{lt}^2 required to satisfy the equation

$$\hat{L}(S) - \textit{loss tolerance} = 0. \tag{4.9}$$

Then, any sample of sufficient size to produce a variance less than or equal to σ_{lt}^2 is acceptable in that it will be likely to produce an attribute choice with full utility. By building a tree in which each attribute has been chosen to satisfy the loss tolerance equation, we are hoping to control both the accuracy and utility of the overall learned result. The ERSS is the smallest such sample size. To solve Equation 4.9, take a small initial sample of a few hundred instances, calculate the parameters $\hat{e}$ and σ^2, then ignoring σ for now , solve by iteration for the $\sigma = \sigma_{lt}$ that satisfies Equation 4.9. We only vary σ in $\hat{L}(S)$ because it is the sole parameter that changes in a fairly predictable manner as the sample size increases. The other parameter, $\hat{e}$, is determined by the ratios $\frac{p_i^+}{|S|}, \frac{n_i^+}{|S|}, \frac{p_i^-}{|S|}$ and

$\frac{n_i^-}{|S|}$. A reasonable choice that can be made here is to assume that as $|S|$ increases, the ratios, and therefore $\hat{e}$ will remain constant. In empirical trials, we used a bisection iteration to solve for σ_{lt}, which turned out to be efficient. To date, it has required no more than 12 iterations to reach a solution.

The last step is to relate the sample size $|S|$ to the variance. Solving for $|S|$ in terms of the variance from the equations derived in Section 4.2.2 is clearly out of the question. The approach taken instead is to plot a curve for the sample size as a function of the variance. A family of second order polynomials was chosen based on extensive empirical testing, and the knowledge that as per Section 3.3.1, the variance is mainly a linearly decreasing function of the sample size. For a particular problem, first the two points $(\sigma^2, |S|)$ and $(\sigma'^2, 2|S|)$ are used to select the curve, then the ERSS is the sample size along this curve that corresponds to σ_{lt}^2. The empirical results show that this calculation of the ERSS meets with good success.

However, the one-shot sampling approach is not adequate; what is needed is a sampling *strategy* that determines an entire sequence of samples to take. ERSS is an estimate of I^*, the optimal number of random samples required to reach the loss tolerance. The argument against the one-shot strategy is that once an ERSS is produced, we have an estimate for I^* based on some incomplete information (information that currently can not support a decision that meets the loss tolerance). If the next sample size is simply the ERSS, the risk is run that ERSS $\gg I^*$ (in fact, we occasionally saw ERSSs of up to 42 million instances, from initial sample sizes of 250). The opposite approach is to increase each sample size by one instance, until the expected loss matches the loss tolerance; but this approach is likely to be very expensive. Somewhere in the middle of these two extremes lies a more rational strategy.

Size of the next sample

The goal is to find a large enough sample S so that Equation 4.9 is satisfied, but to do this with minimal cost (in this case, time). The generic algorithm for this problem has the form:

On iteration k, $k \in 1..n$

1. Select a random sample S_k.
2. Compute $\hat{L}(S)$
3. If $\hat{L}(S) \leq$ *loss tolerance* then stop; else
4. Determine the size $|S_{k+1}|$ for the next step, Goto 1.

The cost of this algorithm is dependent on the size $|S_k|$ of the sample[3]. The cost can be modeled as a linear function of two factors: the overhead time required to process a sample of any size, and the additional time required for each instance in the sample (c and i, respectively). Therefore the cost of one operation (one sample) is $c + |S_k|i$, and the total cost of the algorithm is $nc + \sum_{k=1}^{n} |S_k|i$. Since the minimal cost is $c + I^*i$, we want to find the variables n and $|S_k|$ (for

[3]Note that after computing the summary statistics of the sample, the costs of steps 2, 3, and 4 of the operation are independent of the sample size. The computation of the summary statistics can be considered part of the cost of selecting the sample.

$k \in 1..n$) that minimize[4]

$$nc + \sum_{k=1}^{n} |S_k| i - (c + I^* i).$$

At this point we restrict the general form of the algorithm. What follows is a rather ad-hoc approach to determining a sampling strategy, but the result seems reasonable. The restriction is to force $|S_j| = |S_k| = |S|$ on each iteration, and furthermore to add each sample S_k to the previous samples $S_1 \ldots S_{k-1}$ when computing $\hat{L}(S)$. By combining subsequent samples like this, we are essentially building up the final sample S one chunk at a time. The new form of the minimization is to find the n and $|S|$ that minimize

$$nc + n|S|i - (c + I^* i). \tag{4.10}$$

With this restricted algorithm, the stopping condition tells us that the sample on the last iteration will be at most $|S| - 1$ instances larger than I^*. This means that $0 \leq n|S| - I^* \leq |S| - 1$. If we choose $n|S| - I^* = 0$, then solving Equation 4.10 gives $|S| = ERSS$ which conforms to the one-shot approach. Instead, set $n|S| - I^* = |S| - 1$, giving $n = \frac{|S|-1+I^*}{|S|}$. Substitute this into Equation 4.10, and minimize by setting the derivative to 0. Since I^* is unknown, estimate it with the ERSS.

The result is that the suggested size of the next sample to take is

$$|S| = \sqrt{(ERSS - 1)\frac{c}{i}} \tag{4.11}$$

which is much more conservative than the one-shot approach. Note that $\frac{c}{i}$ is a machine/algorithm specific constant which can be estimated for any system.

The end result is to choose the next size of the sample as a function of the square root of the ERSS, as opposed to immediately generating a sample of size ERSS. In the implementation, we calculate the next sample size based on Equation 4.10, but allow ERSS to change with each iteration (the additional information gives a better estimate of I^*).

4.2.5 Evaluation

There are two hypotheses that could be tested in an empirical evaluation of these methods:

1. the algorithm's answers (decision whether to defer the choice and if so, the estimated sample size required to commit) are sensible on an appropriately wide range of inputs, and

2. when the algorithm is used in the construction of decision trees, the resulting speedup and any loss of accuracy are appropriate to the parameters used.

In this chapter the first and more fundamental hypothesis is tested extensively. The fact that faster choices of attribute (using similar but ad hoc techniques) can lead to speedups of at least an order of magnitude on practical tasks has already been established in previous work [13], and the last example shows the affect of the loss tolerance on the accuracy of the learned decision trees. This section describes the evaluation method used to test the first hypothesis, and presents and discusses the results.

[4]The trivial solution $n = 1$, $|S_1| = I^*$ is not acceptable, it is the one-shot approach.

Method of Evaluation

The evaluation uses synthetic data rather than data from real domains, because we need to know the value of a parameter which is not obtainable on real data: the true difference in information gain (DIG) between two attributes (similar to the δ^* parameter from Section 2.3). The information gain (IG) for an attribute computed from some finite set of examples is only an estimate of the "true" IG. As the sample size increases, the relative frequencies of each combination of class and attribute value converge towards the true joint distribution, so the estimates for IG converge towards the attribute's "true" IG. With synthetic data, we start with the occult knowledge of the true distribution and use it to generate the counts for class and attribute values according to the desired sample size. But it can also be used to calculate the attribute's true IG, along with the DIG between a pair of attributes following two possibly different distributions. The DIG provides a measure of both the difficulty of choosing between the attributes and of the gravity of making the wrong choice. Choosing between a pair of attributes with a large DIG such as 0.1 bits will typically be easy even on a small sample of a hundred examples, but making the wrong choice is likely to yield a less accurate tree.

As a concrete illustration of what could for practical purposes be considered a small value for DIG, we duplicated Quinlan's [78] 14-attribute weather data 512 times and compared the original attribute "outlook" (gain of 0.24675 bits) with an almost identical copy where one of the 7168 examples had been corrupted so as to disadvantage it most (the first example's value was changed from "sunny" to "overcast"). The DIG between the original and the corrupted attribute was 0.00163 bits. In consideration of this, we usually quote figures for DIGs in millibits.

Our experiments present the algorithm with millions of choices between attributes, following a wide variety of sample sizes, distributions (hence DIGs), and loss tolerances. For the sake of anyone who wishes to reproduce the evaluation or to investigate the nature of the set of the distributions used, this paragraph specifies the parameterized procedure that generates the set. Taking the simplest case of two attributes (A and B) with two values each (a^+, a^-, b^+, b^-) and two classes (p and n), a complete joint distribution on these three variables can be specified with the probabilities of eight mutually exclusive and exhaustive events. Each joint distribution is computed from seven numbers representing the following conditional probabilities: $P(p), P(a^+|p), P(b^+|p, a^+)$, $P(b^+|p, a^-), P(a^+|n)$, $P(b^+|n, a^+)$, and $P(b^+|n, a^-)$. The specific values taken for probabilities were $1/2, 1/4, ...1/2^{ndv}$, where the positive integer ndv is the parameter given to the procedure. These values are taken in every combination by the seven conditional probabilities, so the number of distributions it produces is ndv^7. These experiments used $ndv = 4$, generating 16384 distributions. For each combination of 13 values for sample size and 3 for loss tolerance, only a single set of class and attribute value counts was generated; ideally, many should be used.

Choice of commit or defer

The two dependent variables measured in this experiment are the commit rate (the percentage of cases on which the algorithm committed to a choice between the two attributes), and the average loss (the sum of the DIGS for those cases where the attribute with the smaller gain was chosen, divided by the number of cases with a committed decision, whether right or wrong). The distributions are separated into five "buckets," each containing all those with the DIGS within a certain range, as indicated in the graphs of Figure 4.3. The upper number with one place after the decimal point at the center right of each graph gives the upper limit on DIGs in millibits (mb) appearing in the bucket; the lower limit is implicitly the upper limit of the next bucket. The integer below it gives the number of distributions in each bucket. The lowest graph covers distributions where the DIGs are so low that the attributes are identical or practically indistinguishable; its lower limit is zero.

The graphs share the following format. The log scale x-axis shows the size of the samples generated in units of thousands of examples. The linear scale y-axis on the right side of the graph refers to the percentage of samples on which a decision was made (the commit rate). The log scale y-axis on the left indicates the average loss on those samples where a decision was made, in millibits. Values less than 0.01mb are are lumped together as zero. Curves for both commit rate (marked by a "+") and average loss (marked by an "×") are provided for three values of loss tolerance: 4mb (solid), 0.4mb (dashed), and 0.04mb (dotted). Triangles mark the average losses that are sustained by the "strawman" alternative of always committing to the attribute with the larger information gain on the sample.

The broad trends in the graph are as one would expect. The strawman loss curves tend steadily towards zero as sample size is increased. The steeper slope of these curves for the larger DIGs is due to two factors. The average losses tend to be larger on small samples, because the individual losses (DIGs) being averaged are larger, and because the decision is more often wrong on the small sample than on a larger sample. Every commit curve has an S-shape; the "knee" where it ramps up is pushed to the right as DIG decreases (lower buckets) and as loss tolerance decreases (from solid to dashed to dotted curves). Intuitively, this means more examples are required as the alternatives become more similar and as higher accuracy is requested.

The loss curve for a lower loss tolerance is generally lower; a few exceptions occur for very small commit rates where the figure given as average loss may consist of a single case. The solid curve (greatest tolerance, hence least conservative) sustains losses below the strawman's only on the upper three buckets. The dashed curve is more conservative: on the upper buckets, it maintains losses that are roughly an order of magnitude below the strawman losses. Its knee consistently sits around the point where the strawman losses drop below 0.04. The dotted loss curve is the most conservative: its losses are so low on all buckets that they are only occasionally distinguishable from zero. Giving the algorithm a loss tolerance of 40mb or more yielded a 100% commit rate across the board.

It is unreasonable to expect the commit rate to jump to 100% as soon as the average loss in one of the buckets drops below the loss tolerance. The algorithm has the task of keeping the average loss below the loss tolerance over all the distributions given to it that could have produced a given sample (weighted by the probability of that distribution given the sample), not just over distributions with true DIGs within a certain specified range.

Results for the ERSS

The second experiment assesses the accuracy of the algorithm's forecasts for the expected required sample size (ERSS) to commit. To do this we create a new pair of cases from each case on which the algorithm defers, generating more examples[5] from the same distribution as the original

[5]Both the new cases included all the counts from the original cases, as this is consistent with the algorithm's approach. The evaluation could also be done with the new cases generated from independent samples, which may

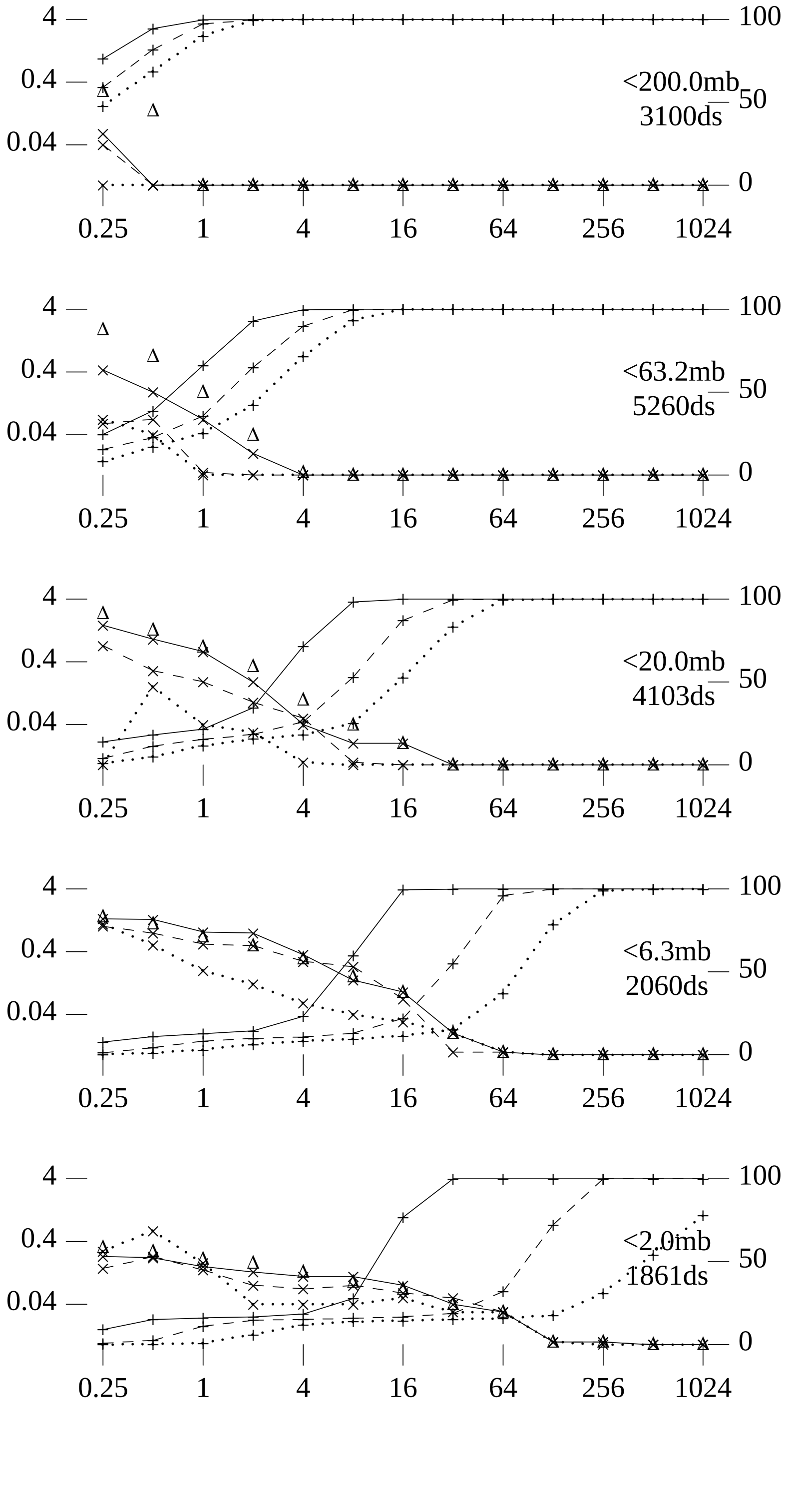


Figure 4.3: **Commit Rates and Average Loss**

For legend, see text in this section.

LT mbits	SS 1000's	NC	Under %	Hit %	Over %	Rev %
4	0.25	11857	36.9	24.5	34.5	4.1
4	0.5	10153	33.0	30.0	32.8	4.2
4	1	8351	27.6	37.7	30.0	4.8
4	2	6362	23.7	44.7	26.8	4.8
4	4	4270	19.0	52.5	23.2	5.3
4	8	2225	15.0	61.8	18.8	4.4
4	16	439	9.6	77.2	10.3	3.0
4	32	0	-	-	-	-
0.4	0.25	14427	41.9	17.2	38.1	2.8
0.4	0.5	13141	39.9	20.0	37.6	2.6
0.4	1	11706	37.1	24.2	35.8	2.9
0.4	2	10179	34.6	28.2	34.5	2.7
0.4	4	8375	32.2	34.6	30.9	2.4
0.4	8	6894	29.2	39.7	28.4	2.7
0.4	16	5358	25.8	47.3	24.7	2.1
0.4	32	4014	21.9	53.9	22.0	2.1
0.4	64	2880	18.0	63.3	16.8	1.9
0.4	128	1973	14.8	68.7	14.8	1.7
0.4	256	1303	12.6	74.0	11.4	2.0
0.4	512	854	8.0	81.4	8.7	2.0
0.4	1024	412	11.2	76.5	10.9	1.5

Table 4.1: **ERSS Accuracy**
Accuracy of estimated required sample size for the algorithm to commit.

case: the number of examples in one of the pair is slightly larger (25%) than the size indicated by the ERSS; in the other it is slightly smaller. Both of these enlarged samples are then given to the algorithm to see whether it commits. We deem the ERSS accurate on the case if the algorithm commits on the larger sample but defers on the smaller one. If it defers on both we deem the ERSS underestimated. If it commits on both we deem it overestimated. The unhappy last combination, where it commits on the smaller sample and defers on the larger one, can never be eradicated because of the stochastic way the cases are generated; we call these results "reversed." Table 4.1 shows the percentages for each of these four combinations, over a variety of loss tolerances and sample sizes. The third column gives the number of cases on which the algorithm deferred.

Hit rates rise monotonically: the greater the sample size, the more accurate the figure for ERSS. For a given sample size, greater accuracy is obtained where loss tolerance is greater. This is very plausible because small loss tolerances require larger samples to commit on, so the correct figure for ERSS becomes a more distant target. At all times the percentages for under- and over-estimates are approximately equal, as we would want.

better suit other methods.

For practical purposes, the important conclusion to be drawn from this table is that after a few thousand examples, the majority of the estimates of the required sample size are fairly accurate (+/-25%), and that the algorithm behaves sensibly and smoothly over the relevant range of parameters. For a wider tolerance (+/-50%), the majority is reached at 500 examples for 4mbits and 2000 for 0.4 mbits.

Utility of Trees

One thing that has been implicitly assumed in this chapter is that the utility or quality of the tree will be positively correlated to the reduction of the loss of information gain at each point in the tree construction. This section uses the knight-rook-vs-king data from Quinlan as the database, and shows the learning curves of the decision trees produced as the loss tolerance is varied from very small to very large. The purpose of this example is to show graphically that a strong connection exists between the loss tolerance at an attribute and the overall prediction accuracy of the learned tree.

Figure 4.4 shows several different learning curves for decision trees built using loss tolerances varying from .001 to .04. For the knight-rook-vs-king data set this range is broad enough to show average sample sizes from about 10,000 instances (loss tolerance = .001), to about 10 (loss tolerance = .04 - .015) being required before decisions can be made.

There are several ways to build a decision tree using the loss tolerance information based on what happens at a node when it is determined that more samples are needed. Do these samples get added to the current set of samples at that node? We have chosen not to do this in order to make a point. What we have done is to take the recommended sample and use that to make a judgement at that node, or decide that more samples are needed. When there is enough information to make a judgement, we throw the additional samples away and proceed to the next decision. This has the affect of limiting the size of the trees by the number of samples originally given to the tree. The learning curves above represent trees built on 50, 100, 200, and 400 samples. What has turned out to be interesting about this is that the effectiveness of the information gain measure is greater on smaller trees than on larger. This is seen in the figure by noticing the decreasing slopes of the curves as the original sample size grows. This makes sense, because the more complete the tree gets, the less it matters (in terms of accuracy) whether or not correct decisions were made at the top.

The learning curves show clearly that decreasing the loss tolerance tends to give higher prediction rates for decision trees, as expected. In general, information gain is not a perfect measure of the quality of a tree, but it has performed well in the past on a wide variety of domains. The culmination of this work would be to derive a mathematical relationship between the loss tolerance at a node, and the expected quality of a tree, but this quantity would be a very difficult one to produce.

4.3 Application to Belief Networks

When constructing the decision tree, the possible actions in the two action decision problem are to select either attribute A_i or A_j for building the next node of the tree. The sample is to be large enough either to be able to distinguish which attribute is the better choice, or to determine

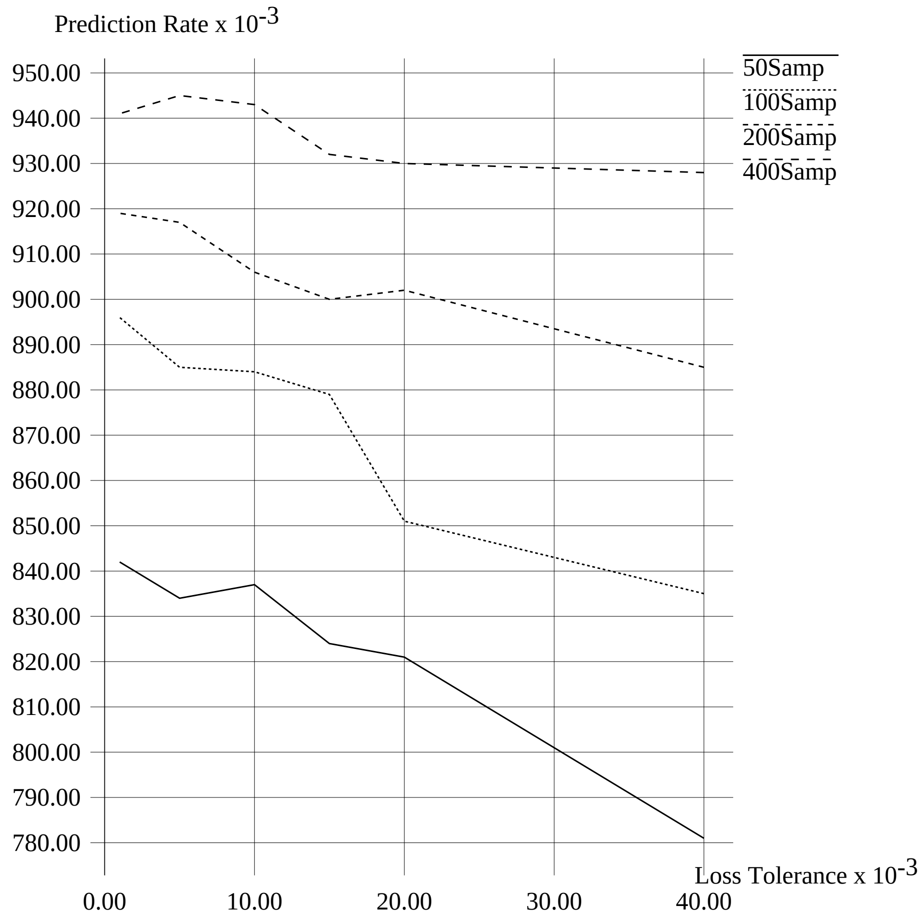


Figure 4.4: **Learning Curve Versus Loss Tolerance**
The prediction rate of decision trees learned using the loss tolerances shown. The curves represent different size trees being learned.

that either attribute can be chosen since they are almost exactly the same. When constructing a belief network, or more precisely when learning the CPTs of the belief network given the graphical structure, the problem changes a little.

In fact, there are two types of situations under which one would want to control the sample size when constructing belief networks. The first situation is when the inferences from the network must be of a certain specified accuracy, so the sampling is controlled to provide at least that required amount. The second case is more in line with the two-action decision problems of the previous section. It is when there are multiple models for a given domain, each recommending a course of action, and the task is to determine which of these actions should actually be taken.

4.3.1 Inference Accuracy

For the first case, the problem is not a two-action decision problem; there is no decision to make besides determining how much information to collect before accepting the results. Nonetheless, it is highly desirable in some cases to be able to determine on the fly the sample size likely to be needed in order to reach a particular degree of accuracy for a particular inference. This capability is useful in any situation in which there is time pressure, and in which new models must be constructed on a fairly regular basis. This happens to be just about every situation that humans deal with. Consider for example how one processes visual data when opening a door and walking into a room; typically a person will open the door, scan the inside of the room for about a second and see if there is anyone inside that is interesting, then go about his/her own business. It would be silly to spend ten minutes examining every crack in the ceiling and every hair on the floor every time we wanted to change our location.

Confidence intervals are common statistical tools for providing a degree of confidence in results. The statement $Pr(a \leq x \leq b) \approx .95$ represents a 95% nominal confidence interval about x. The correct interpretation of this statement is: the confidence interval procedure that generated the interval $[a, b]$ has probability .95 of generating an interval that contains x. The rest of this section uses confidence intervals as the vehicle for specifying the desired accuracy of inferences. The connection between the ERSS and a given confidence interval is defined below.

First the mathematics are presented, then we will go into an example in detail. It is known from Chapter 3 that every inference made from belief nets induced by sampling with replacement has some beta distribution $\beta(\alpha, \omega)$. By the mean limit theorem, we know that for α and $\omega \gg 1$ then $\frac{\alpha}{\alpha+\omega}$ is approximately normally distributed. Statistical theory then shows that the nominal $100(1-\theta)$ confidence bound is found as:

$$Pr(\hat{\mu} - z_{1-\theta/2} SE(\hat{\mu}) < \mu < \hat{\mu} + z_{1-\theta/2} SE(\hat{\mu})) \approx 1 - \theta, \qquad n \gg 1 \tag{4.12}$$

where $z_{1-\theta/2}$ is the $1 - \theta/2$ quartile of the unit normal distribution $N(0,1)$, μ is the desired probability, $\hat{\mu}$ is the mean of the beta or $\alpha/(\alpha+\omega)$, and $SE(\hat{\mu})$ is the standard error of $\hat{\mu}$, which equals $\sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (\mu_i - \hat{\mu})^2}$. Another form for the standard error is in terms of the sample variance, which we can approximate with the variance of the beta distribution:

$$\begin{aligned} SE(\hat{\mu}) &= S/\sqrt{n} \approx \sqrt{\frac{1}{n} Var(\beta(\alpha, \omega))} \\ &= \sqrt{\frac{\alpha\omega}{(\alpha+\omega)^2(\alpha+\omega+1)^2}} \\ &= \sqrt{\frac{\alpha}{\alpha+\omega} \frac{\omega}{\alpha+\omega} \frac{1}{(\alpha+\omega+1)^2}} \\ &= \frac{\sqrt{\hat{\mu}(1-\hat{\mu})}}{\alpha+\omega+1}. \end{aligned} \tag{4.13}$$

Therefore, to achieve 95% nominal confidence interval of $p \pm .03$, set $z_{.975} SE(\hat{\mu}) = z_{.975} \frac{\sqrt{\hat{\mu}(1-\hat{\mu})}}{\alpha+\omega+1}$ equal to .03, then solve for

$$n + 1 = \alpha + \omega + 1 = \frac{z_{.975}}{.03\sqrt{\hat{\mu}(1-\hat{\mu})}}. \tag{4.14}$$

We will use the dog-out example from Chapter 3 to make the ideas clear. This example is a bit poor in some ways though, for instance it does not have the element of time pressure, nor is there a pre-existing large database of examples for Mr. Smith to draw from. Let the goal be to sample enough to find the probability that the dog is out to within $\pm.03$ with a *confidence* of 95%.

Consider the problem of finding the probability of $Pr(do|fo, bp)$ (dog out given family out and bowel problems). The first step has already been done, which is to build the model on the basis of a sample large enough to provide some useful information. Next, we need to find the number of additional samples required to produce a 95% nominal confidence interval of $\pm.03$ around the target probability. We know from Table 3.5 that the current distribution on the basis of the 100 samples already taken is $\beta(6, 1)$. The total number of *relevant* samples required is found as above by solving Equation 4.14, thus giving an n of about 187. Since the first 100 samples for the entire network gave only $6 + 1 - 2 = 5$ relevant samples (subtract 2 for the prior of $\alpha, \omega = 1$), then to get an additional 182 relevant samples, one would expect to have to take about 3600 more samples. Poor Mr. Smith will be working on this table for a long time, the main reason being that the combination fo and bp is very rare. If a more common probability such as $Pr(lo|\mathbf{fo})$ (outside lights off, family not home) is desired, then as there are already 68 relevant samples for that distribution, Mr. Smith needs only take 175 more samples.

The inference can also be spread across many tables, for example $Pr(\mathbf{bp}|\mathbf{fo})$ (the dog mysteriously develops bowel problems whenever the family is out). It is almost as simple. The only extra step is to use the information already available in the network and compute the inference distribution $\beta(\alpha, \omega)$ as per the transformations given in Chapter 3. Then just as done above, compare the number of relevant samples already seen $(\alpha + \beta)$ to the number of relevant samples that must be taken and from this determine the total number of new samples to take.

There is another option which we have not explored at all. To date, the database has been viewed as an entity for which sampling entails randomly picking a complete sample that includes all the variables in the database. This process does not give one the ability to specify what characteristics the sample will have. This is a bit unrealistic when you consider the capabilities that any database manager from 20 years ago can supply. For example, assuming for a moment that there is a large database in the background, then in SQL one could ask for all of the data, and only the data, that pertains to the weak column fo, bp (a column with very little data). By sampling like this, instead of creating a general sample of which only a small percent of the instances are relevant to the probability in question, one can instead draw a fraction of the original sample, all instances within which are relevant. This has not been explored yet for many reasons, not the least of which is that the success of such sampling is highly dependent on the database manager in question, the particular entity relationship structure of the database, and the way in which the index tables are set up.

4.3.2 Multiple Models

Consider market prediction on Wall Street, a complex domain with heavy time pressure and the need for correct answers. When modeling a complex domain, there are usually many ways in which a model can be built, and it is often unclear which is superior. Furthermore, when the likelihood of data entry error is high, or in an environment that is rife with competition, rivalry and secrecy, it is common, or even necessary to have several sources for important information. What this leads to is that a user might be faced with the situation where he has two belief network models of some task which are producing different results. Which model to believe? Let M_1 and M_2 be two distinct models which are returning two distributions D_1 and D_2 for the probability of success X_1 and X_2 of two possible actions A_1 and A_2. The task is to determine the maximum utility sample size to use in collecting enough information to distinguish these actions.

The mapping from the decision tree solutions into this problem is basically one to one, so we provide only a quick, concise description of the process, and refer the reader to Section 4.2. From Chapter 3 it is known that D_1 and D_2 will be two specifiable beta distributions, $\beta(\alpha_1, \omega_1)$ and $\beta(\alpha_2, \omega_2)$. With the decision tree, we estimate the goodness of the action (the attribute selection) by the expected loss of the information gain of the attribute as in Equation 4.7. For belief networks, there is no commonly accepted measure of goodness that matches the information gain measure in decision tree theory. But, there is an obvious choice here, which is simply to use the inference distributions themselves in the loss equation. The more information present, the narrower the distributions. The loss itself is again the difference $X_2 - X_1$ times the probability of that difference occurring, giving

$$L(S) = \int_{X_1=0}^{1} \int_{X_2=X_1}^{1} (X_2 - X_1) f_2^S(X_2) f_1^S(X_1) dX_2 dX_1. \tag{4.15}$$

Approximate the betas with normals and let $y = X_2 - X_1$ be the normal $N(\hat{e} = \hat{e_2} - \hat{e_1}, \sigma = \sqrt{\sigma_1^2 + \sigma_2^2})$ where $\hat{e_1}, \hat{e_2}$ and σ_1^2, σ_2^2 are the means and standard deviations of $B(\alpha_1, \omega_1)$ and $B(\alpha_2, \omega_2)$ respectively. The loss equation again is

$$\hat{L}(S) \quad = \quad \frac{\sigma}{\sqrt{2\pi}}(e^{-\hat{e}^2/2\sigma^2} - e^{-(1-\hat{e}^2)/2\sigma^2}) + \frac{\hat{e}}{2}(erf(\frac{1-\hat{e}}{\sqrt{2}\sigma}) + erf(\frac{\hat{e}}{\sqrt{2}\sigma})).$$

Set $\hat{L}(S)-$ *loss tolerance* $= 0$, and as in Section 4.2.4 solve in terms of the statistics $\alpha_1, \alpha_2, \omega_1$ and ω_2, thus giving the desired sample sizes. The user must again be able or willing to specify a loss tolerance.

4.4 Summary

This chapter raised four questions important to large-scale induction: how to decide whether the better action can be confidently chosen on a given subsample, how to estimate the size of the subsample required to do so, how much this choice is likely to cost in terms quality, and how to choose the sequence of subsamples to achieve this most efficiently. The results in this chapter are useful for characterizing and controlling the sample sizes required to produce induced models of specifiable accuracy. The theory and detailed methods to address these issues were presented, along with an empirical evaluation of most of them.

Chapter 5

Improving Learning

Section 3.2 examined an approach to learning CPTs called bookkeeping, where updating a statistical summary of samples relevant to each cell is the basis of the learning. Chapter 3 further showed how this approach leads to the specification of sets of beta distributions that can be used to provide inference distributions for any inference in the network. But, we also showed in Section 1.3.2 that this simplistic approach is often not good enough, requiring too much data to be very successful. The purpose of this chapter is to improve the learning process by attaching user-specified learning elements to each CPT, then integrating the output from these algorithms back into a general belief network. Results of an implementation of these ideas are shown. The last section discusses learning with hidden nodes in the network.

5.1 Motivation

As discussed in Section 1.3.2, there is one particular commonly occurring situation that requires us to supplement the statistical learning described in Chapter 3. This is when the CPT's are sparsely populated by samples. Recapping from the introduction, the combination of huge CPTs and the propensity to have an uneven distribution of the probability mass over the unique parent instantiations gives way to low-probability columns. Even if the database is very large, it will be relatively easy to find CPTs that have columns that have seen little data, and thus the probabilities in that column are largely undetermined. If we think in terms of representing and learning a function from the parents of a node to the conditional probabilities in a node, it immediately becomes evident that the statistical induction approach is equivalent to the table-learning methods of generalization [83]. Table learning methods were put to pasture decades ago because they do not have any generalization power. The power to generalize is the power to learn, without this ability a system will never be able to extend past what has already been explicitly seen.

The solution is to apply stronger learning algorithms to the task. A neural network (for example) will generalize from the data in the data-rich columns to supplement the sparse information found in data-poor columns, often to great advantage.

The ability to apply general learning algorithms to the task of CPT learning not only allows stronger generalization, but also provides the ability to take advantage of the domain specific information that exists in every application. For example, there might be several tables in which a linear relationship between the node variable and the parents is expected, others where the tables are narrow (few unique parent instantiations) and well behaved (equal probability distribution across parent values), and yet other cases where many of the columns in the tables will be sparse. What makes sense then is to allow the user to specify a different learning method for each table. The tables that are expecting a linear relationship are learned with linear regression techniques. The tables with a data overload can be learned with the bookkeeping approach and the tables that have sparse data can be learned with a neural network or a decision tree. This ability to tailor the choice of learning algorithm to the situation at hand can make a tremendous difference in the effectiveness of the overall approach.

In fact, we take this one step further. In tables with a mixture of data-rich and data-poor columns, what we do is first send the data off to a user-specified learning algorithm. When the process is finished, a separate process fills out the CPT of the belief network using the bookkeeping approach for columns with a statistically significant population, and the results from the selected learning algorithm to fill out the sparse columns of the table. The advantage of combining the two approaches within one table is that for the data-rich columns the beta distributions are still available, while at the same time the data-poor columns have more accurate results.

The results have been quite excellent in the car insurance domain that has been used, but not without a cost. The price that has been paid is that we are back to learning point probabilities in the CPTs. The reason is that we are allowing any general learning algorithm to be applied to the task of learning the CPTs; what is the output distribution of, for example, a neural network? Without beta distributions in the CPTs, the distribution propagation theorems described in Section 3.4 no longer apply. If, however, we are willing to approximate the new cell distributions with betas, then we would get the benefit of more accurate predictions along with the ability to return inference distributions. This would introduce a major assumption into the works, but that assumption may be more aesthetic than not given that the learning is providing on the order of 25% to 60% gains in accuracy.

One might argue that doing any extra work to get better prediction out of data-poor columns is silly. If the data in a column is sparse, then the odds of needing information about that column are small, and so the improvement is not worth much. This not entirely correct. The real situation might actually be the reverse of this line of thought, as the data that is hard to find, or expensive to produce will be the sparse data, and *sometimes* it will be just that data which is the most useful to the user. Consider, for example, a medical database that is being used in part to predict the success rate of a new cancer drug. The cost of each test case might be a human life (or if the drug is successful, then it might be the several people that could have taken the drug during the testing phase). The cancer data will be sparse, but the results are of critical importance.

5.2 Applying Neural Nets

A neural network is an iterative supervised learning approach that minimizes an error function by gradient descent. The training phase of the neural network is split up into n *epochs* or iterations. Within each epoch, the data is applied one instance at a time to the inputs of network. For each datum, the output of the output unit is compared to the true value that the unit takes in the data, and a function of that difference is propagated back through the network in such a way that on the next iteration, the difference will be smaller. The training phase is run until the network

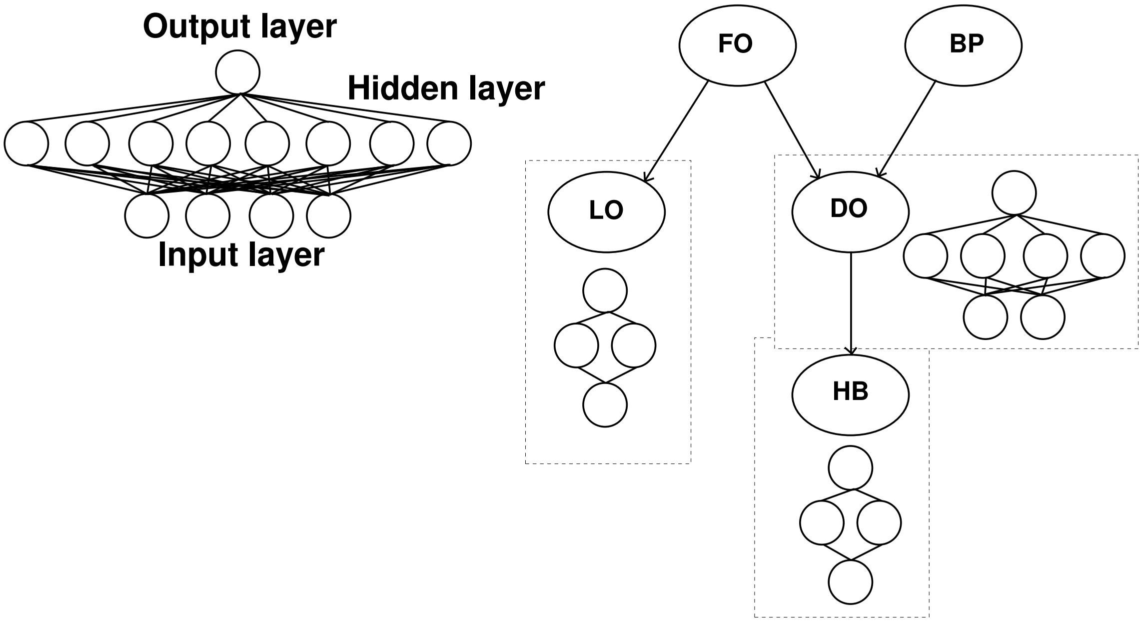


Figure 5.1: **Divide and Conquer**

The figure on the left is a neural network applied to the dog-out problem, with the task of predicting whether or not the dog is out (do), based on the other four variables in the system. On the right is the belief network corresponding to the same problem. The neural net corresponding to each CPT is shown next to the node.

converges to a stationary point, or until the time allotted for the training phase has run out. In general, it is difficult to place good bounds on the convergence properties of neural networks.

Beyond the justifications given above, the integration of belief networks and neural networks has several attractive properties. Directly applying a single neural net to a real problem has several shortcomings. The large size of the domain leads to large networks with very complex modeling tasks, and the result is the ability to predict only one function over the entire system of variables. The structure that the belief network lends to the integrated model imparts the ability to explain inferences, and to produce any inference over the set of variables in the network. Furthermore, the belief net reduces the task into many smaller, simpler, localized modeling problems. The neural nets applied to these smaller problems are likely to be much smaller in size, and due to the decreased complexity of the task, are likely to be more accurate. Figure 5.1 shows an example of this showing the two approaches for predicting whether the dog is out. The neural network on the left has a total of 13 units and 40 arcs. In the belief network the top two nodes have no parents so they are learned by bookkeeping. The neural networks for the other three nodes are shown next to the nodes, the largest of them having 7 units and 12 arcs. The reduction in complexity becomes much more significant as the problems grow more realistic. This section will discuss the more interesting of the research and implementation details.

The job of the neural network is to produce a density function for each column of the CPTs that it is learning, without any predetermined requirements on the family of functions that could be discovered. Let T_i be the CPT for node X_i, and π_{ij} be a unique parent instantiation for the node. The general process is as follows:

for each T_i that is being learned with neural nets,

1. Retrieve from the database the instances that are relevant to T_i.
2. Construct a neural network NN_{T_i} and initialize it.
3. Train NN_{T_i} on the samples, where the value of the node X_i serves as the classification of each datum.
4. Apply an input combination (π_{i1}) to the network and read off the distribution for the column $Pr(X_i|\Pi_1)$ from the output units.
5. Normalize the output and write the values into the CPT for the belief network.
6. Repeat steps 4 and 5 for all input combinations π_{i2} through π_{im}.

Note there is a substantial difference in the neural network output and the output from bookkeeping. Bookkeeping provides a distribution for each cell in the CPT. The neural net gives a probability function for each column, which is quite a bit less information.

The network being used is a 3-layer feed forward network that is set up to learn density functions (Figure 5.1 shows an example for the dog-out domain). The range of input values to the network is from -1 to 1. The network incorporates ideas of momentum, adaptive parameters, and a few other tricks that are explained below. We can build a network for any combination of input/output variables, including nodes with binary, discrete, continuous, and nominal values.

5.2.1 Constructing the Neural Net

There are four cases for mapping the belief network variables into neural network units, corresponding to whether the variables have binary, continuous, discrete or nominal values. We go over each case below.

Input Units

The input units on the NN (neural net) can have values ranging from -1 to 1. They represent the parents Π_i of X_i in the belief net. The four cases are:

- **Binary:** A belief network with binary variables is mapped such that 0 becomes -1, and 1 remains unchanged.
- **Continuous:** If the continuous range is from a to b, then that range is mapped into -1 to 1 by taking the new value to be: $\frac{2*value}{b-a} - 1$.
- **Discrete:** The discrete case is a bit more interesting. Assume that the BN node being mapped has three values; 0, 1 and 2. There are two different mappings that one could make a case for, namely the *mid-range* mapping and the *max-distance* mapping. The goal in the max-distance scheme is to assign values in the range [-1,1] so as to maximize the distance between the discrete cases. The principle behind this scheme is to try to maximize the separation of cases in the network to allow maximum discrimination. The mid-range method, on the other hand, visualizes the NN range [-1,1] as being split into n buckets, n equal to the number or

discrete values. According to this, the midpoint of that bucket should be returned as the value for the corresponding discrete case. For example, the discrete values [0,1,2] map to [-2/3, 0, 2/3] with the mid-range approach, while the max-distance mapping gives [-1, 0, 1]. We have chosen to use the mid-range mapping.

- **Nominal:** The difficulty with nominal values is that there is no sensible way to order them. If 0 represented "bird" and 1 represented "skunk" then what would .5 be? The only recourse here is to split up the nominal valued nodes into a set of binary nodes, one binary node for each nominal value. The binary nodes can be directly mapped to input units as discussed above.

Output Units

The situation changes when mapping X_i, the node for which the neural net is being built. Each output unit on the NN will output a value between [-1,1] that must be mapped into the probability of that proposition. Thus, we need an output unit for each of the values of X_i, representing the proposition that that value is true. There are two exceptions here, binary and continuous nodes. The binary node can be mapped directly into one output unit. For continuous variables, if X_i was originally continuous, it has already been discretized for the belief network, so we treat it here as being discrete. Thus, for any node X_i, the number of output units will be the number of instantiations of X_i, or $|D_{X_i}|$.

Size

Lets compare the size of the neural net versus the size of the CPT itself. If $|D_{X_i}|$ represents the number of possible instantiations of X_i, then a CPT has $|D_{X_i}||D_{\Pi_i}|$ cells, or $|D_{\Pi_i}|$ columns to be learned. Following our construction, a NN has #input-units $\times$ #hidden-units + #hidden-units $\times$ #output-units weights, along with the hidden and output units, all of which must be learned. Let $|\Pi_i|$ represent the number of parents. There are $|\Pi_i|$ input units, $2|\Pi_i|$ hidden units and $|D_{X_i}| - 1$ output units. Given this, there will be a total of $2|\Pi_i|^2 + 2|\Pi_i|(|D_{X_i}| - 1)$ weights and $2|\Pi_i| + |D_{X_i}| - 1$ units to learn. To put this in more understandable terms, if all variables have 5 values, and X_i has four parents, then the CPT has 3125 cells or 625 columns to learn. The NN, on the other hand, need only learn 76 parameters.

Note that this particular construction uses a relatively small hidden layer. This is the size of the hidden layer that we have used consistently for generating the results, and although it is small, the results from this network have been quite acceptable. One thing that should be made clear is that we are *not* trying to promote this *particular* network construction as the best for the problem of CPT learning. In fact, we will not be making that claim for any of the algorithms proposed here. Our claim is that the *application of* more sophisticated *learning techniques* to this problem is worthwhile.

5.2.2 Update Equations

Let O represent the actual output of the NN, and let O_i^u represent the actual output of unit i on a specific input pattern u. Also, let ζ represent the target output pattern.

The error measure, or cost function E is the quantity being minimized by gradient descent with every training epoch. We selected the following error function with the goal of learning probabilities:

$$E = \sum_{iu}(\frac{1}{2}(1+\zeta_i^u)log\frac{1+\zeta_i^u}{1+O_i^u} + \frac{1}{2}(1-\zeta_i^u)log\frac{1-\zeta_i^u}{1-O_i^u}). \tag{5.1}$$

Using this error function, the probability that the hypothesis represented by unit i is true is $1/2(1+O_i^u)$, (the straightforward mapping of the [-1,1] output into a 0/1 probability).

This choice of error function affects the weight updating procedure a bit. The typical change in the weight between two units i and j is

$$\nabla W_{ij} = \eta \sum_u \delta_i^u V_j^u \tag{5.2}$$

where V is the intermediate output of the hidden unit j, and delta is a function of the difference between the desired output pattern ζ and the actual output pattern O. The modification of the error function has lead to a new δ_i^u term:

$$\delta_i^u = (\zeta_i^u - O_i^u). \tag{5.3}$$

The rest of the update equations remain the same standard equations used for updating multilayer feed forward networks. They can be found in Hertz et. al. [44], or just about any introductory neural network book that describes feed forward networks.

5.2.3 Extensions

We applied three rather standard tricks to speed up the learning process: momentum, adaptive parameters, and memorization.

The idea behind momentum is that gradient descent methods tend to behave poorly when the cost surface of the error function has steep valleys with shallow valley floors that lead to the minimum. In this situation, the gradient descent will tend to zig-zag back and forth across the valley like a ball bearing dropped into a teacup. If the learning parameter η is too small, then the descent will be a very gradual process; if too large, then there will be wild swings in the approach. By giving each weight some momentum, it will be more likely to change in the direction of the average downhill force. This is done by adding a fraction of the previous change in weight to the current change in weight:

$$\nabla W_{ij}(t+1) = \eta \sum_u \delta_i^u V_j^u + \alpha \nabla W_{ij}(t). \tag{5.4}$$

We have set α to .5, substantially less than the .9 that is normally recommended.

When running the network, there was a tendency for very slow convergence. We decided to use adaptive parameters to combat this problem, with some moderate success. The idea is that when the energy function has been decreasing for the past 30 epochs, say, the indication is that we are on a relatively easy portion of the cost surface. If the learning parameter η is increased during this downward trend, we would expect to converge that much faster. Likewise, if the energy function is increasing, then the process might have overshot the valley and thus it would make sense to decrease η. We chose $\eta = .04$ to start with, and set each upward increment at .01. When the energy function increases continuously for 30 epochs, we cut the current η in half.

The last trick we applied to the network actually showed the largest improvements. While training the network, at the end of every epoch the energy is computed. The network that generated the smallest energy was saved, and replaced any time a later epoch provides a network with an even lower energy function. At the end of the allotted time, if convergence was not achieved then the minimum energy network over the entire run is returned.

5.2.4 Network Output

The last question to address in applying the neural networks is: once the network has been trained, what should be done with it? As explained above, the neural net has $|D_{X_i}|$ output nodes. To fill in the CPT for X_i, we apply a unique parent instantiation of Π_i to the neural network inputs, compute the network outputs, compute the probability that each output unit represents, normalize, then fill in that column of the table.

5.3 Applying Other Learning Algorithms

We have also integrated nonlinear regression and decision tree learning techniques into the belief network to show a proof of concept. A brief description of each implementation follows below.

5.3.1 Decision trees

A decision tree algorithm [12, 78, 79] was implemented and tested. The construction is fairly simple. The variables in Π_i become the attributes that the tree is built from. The values of the node X_i are the classes that are being determined. Each column of the conditional probability table is determined by finding the leaf indicated by π_{ij}, counting the samples in that leaf that belong to each value of X_i, then normalizing.

One of the major benefits, normally, of a decision tree is its ability to ignore irrelevant attributes when splitting the data. This benefit is toned down in a belief network, as the graphical structure has already provided much of the relevancy information. But, there are many cases where *certain values* of the node variable X_i will be dependent on a particular parent $A \in \Pi_i$, but the rest of the values in X_i will be independent of the parent. The decision tree can take advantage of this by pruning the tree according to the chi-square test of stochastic independence. The purpose of this test is to measure the likelyhood that an attribute splits a set of examples in some non-random fashion. If the chi-square measure indicates that the attribute is not relevant to the set of examples, that attribute is pruned.

The principle behind the test is that for a random distribution over a set of $m = |D_{X_i}|$ values, the expression $\sum_{j=1}^{m} \frac{(n_j - E(n_j))^2}{E(n_j)}$ will have a chi-squared distribution, where n_j is the number of samples in the current leaf of the tree such that $X_i = x_{ij}$ and $E(n_j)$ is the expected number of those samples. Since this term is easily computable, we can generate the value of this term for every association of attribute and examples in the current leaf of the tree. Given this number, we can determine a confidence level in the belief that this number comes from a chi-squared distribution.

If we believe it, then we believe the attribute is irrelevant to the set of examples, and prune it. We require a confidence of 99% to prune an attribute. Since the 99th quantile of the chi-squared distribution is at about 6.64, we prune any pair whose chi-square measure is less than 6.64.

The DT proved to be a fast approach with results generally on par with the neural network. The algorithm used in the implementation can only deal with binary network variables.

5.3.2 Nonlinear Regression

The non-linear regression algorithm that we used is Breiman's PIMPLE [9], which tries to find clusters of spline functions that describe the input. This method combines forward selection and backward elimination techniques in model selection. Forward selection is a process of determining (or limiting) the maximum degree polynomial that will be used by the regression. Backward elimination is the process of eliminating the highest degree predictors used in the regression *during* the regression (a predictor is a function of variables used in the prediction of the node variable). Typically the combination works well when the underlying function can be approximated by a few terms with a large number of factors (normally, smooth functions without sharp local structures).

The application, in theory, is trivial. The data is directly fed into PIMPLE, and the results are produced in a form that, after some reorganizing, can be written into a CPT. The main problem with applying this particular algorithm to these databases is that it requires that its predictors have at least 4 distinct values. The principle is that the prediction problem is "easy" if there are fewer unique values and thus linear regression or some other such technique should be applied. But this fails to take into account cases where there are several predictors, some of which are very complex, but one or two happen to be binary. By maintaining that none of the parent variables can binary or low-arity discrete variables, the potential range of application of this particular technique is limited. PIMPLE did work well when the input variables and data where diverse enough to allow the algorithm to function.

Though this particular application has a rather limited range of application in general, any linear [85, 84] or nonlinear regression algorithm can be fit into the belief network structure with minimal effort.

5.4 Results and Comparisons

The goal in this section is to demonstrate the effectiveness of applying a variety of learning algorithms to the task of learning the CPTs. We also want to show where these algorithms tend to be most effective, and how best to use them in an integrated system.

Extensive comparative testing between the neural network, the decision tree, and PIMPLE was not done; it is a sensible boundary on this dissertation, which is simply not about optimizing neural network learning, or comparing neural networks to decision trees. The thrust of this chapter is to make the point that other learning algorithms besides statistical bookkeeping have an important place in belief network induction, and to emphasize that point with practical successes.

Five approaches were implemented, the bookkeeping or *STAT* approach, the neural network *NN*, the decision tree *DT*, PIMPLE *NLR*, and a combination *COMB* of STAT and NN. Examples of the learning performance are given for each of these methods, then a closer comparison is made between the NN and STAT approaches. It is important to note that these algorithms are not meant to be the only or the best possible algorithms of this class that can be applied to the task, each is simply representative of a class of general approaches to learning.

$COMB$ is probably the most interesting approach tested. We talked before about the ability to combine *within one CPT* both STAT and NN learning. The combination is done by choosing to fill some columns in with the STAT results and the others with the NN results. Specifically, columns with 20 samples or less are learned with NN, columns with more are learned with STAT. The reason we combine the results on a column by column basis instead of cell by cell is clear when we recall that with the STAT approach, any one sample is relevant to exactly one *column* (see Section 1.3.2), meaning that we have data-poor columns as opposed to just having data-poor cells. The results are quite impressive, but the visible results here are not the only benefit. The major attraction of this approach is that the accuracy of the data-poor columns is substantially improved, but the data-rich columns still have access to their beta distribution representations, which are much more useful for inference.

There have been three databases used in the testing phase; the alarm database from Pearl [73], the dog-out database shown in Chapter 3, and a realistic, moderately sized car insurance database. A small description of each follows in the relevant sections below. The usage of the databases is atypical. Since we wish to compare the accuracy of each of the three learning techniques to STAT, an objective way to determine model error rates is needed. The most straightforward way to do this is to know the model that we are trying to learn. Each database then is actually *generated* in a way that is very similar to the Monte Carlo simulation techniques. We create "The" belief network B_S that represents the actual process occurring in the real world. This network is then used to generate a large database (typically on the order of one hundred thousand samples for each) which are used as the database that is seen by STAT, NN, DT, and NLR. A belief network B_D is built for each approach using a random sample from the big database. This learned belief net B_D is then compared to "The" belief net B_S, and from this point a model evaluation can be made on the basis of hard evidence.

The next issue that must be addressed is: how are the models to be compared? There is a wide range of standard model selection techniques, including Cp, Sp, MSE, AIC, log scoring and so on (see Lauritzen et. al. [53]), but none of these need be used. We have a big advantage– we know the right answer. To score our models, we simply take the absolute value of the difference between the predicted probability and the correct probability, and average that over all the cells in all of the CPTs that are being learned. This is the *mean error*. The method with the lowest mean error is the method that wins. We have other ways of analyzing exactly where the improvements have come from; these will become clear in later sections. Note that one implication of the mean error scoring system is that a data-rich column counts the same as a data-poor column. This is intentional, as it is often the case that a good description of the data that is easy to get (data-rich columns) is not worth as much as a good description of the data that is hard to get (data-poor columns). This approach is incorrect in that utility theory is not being used (as is any approach that does not tie utility theory into the evaluation). For example, in some domains it would make more sense to weight the error in the columns by the probability of that column occurring (meaning the data-heavy columns would count more). We show an example with this metric in Section 5.4.3. The point is that without access to domain knowledge that tells us which columns are the important ones, then it is not clear which approach is better.

The experiments done over the three databases mentioned above are shown next.

5.4.1 Burglar Alarm Database

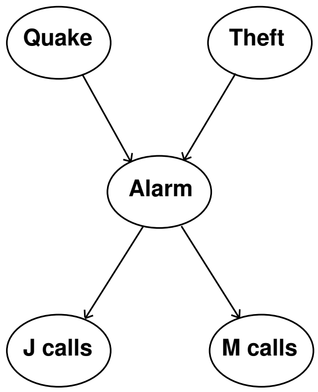


Figure 5.2: **Burglar Alarm Belief Net**
This is a binary belief network, where the nodes represent an earthquake, a burglary, an alarm going off, John calling and Mary calling.

This section gives results from the burglar alarm database. The belief network that generated this database comes from the graphical structure depicted by Figure 5.2. We show the CPTs for this and the other networks in the Appendix. This is a common example that we used as reference point due to its popularity in the literature. This database is also a good example of how rare data can often be the most valuable in the database, as in this case what we are likely to care the most about are the probabilities concerning earthquakes and burglaries, both of which are pretty rare events.

All of the data in this chapter is the result of running the following process 10 times: generate a random sample of the given size, pass the sample to three separate processes STAT, NN and DT and have them learn the CPTs, then generate the error measurements.

The first set of results shows a comparison between STAT, NN and DT on sample sizes of 100 and 500. NN was run for 20 epochs (NN was essentially converging at this point, as this network is very simple). Table 5.1 shows the mean error comparisons for the algorithms on the 100 sample and 500 sample data sets. Note that for this simple database, NN consistently produces networks with a lower mean error. For the 100 sample case, it beats STAT by over 50%, and DT by over 20%. In the 500 sample case, it displays even better results, outpredicting STAT by more than 60%. This is odd in that we expect STAT to do better on larger data sets. The better performance of NN here is probably due to the simplistic nature of the problem. The standard deviation of the results is given as well; note that STAT generally has the least deviation in its error, indicating a more consistent approach in general. The other two are more often correct, but when they are wrong, they will tend to be more wrong than STAT.

To get a better feeling for where and how the improvements are taking place, we characterize the mean error of classes of columns in the learned CPTs, where the classes are determined by the relative sparseness of the data in the columns. The two graphs below should be interpreted as follows: The X axis represents the columns in the CPTs that have seen ≤ 10 samples, 10 to 20 samples, and so on, up to the last point on the axis which represents all of the rest of the columns, all of which are data-rich. The Y axis is the mean error over those groupings. Note that the general

100 Samples		
Approach	Mean	Standard Dev.
NN	.0788	.038
DT	.101	.05
STAT	.161	.021
500 Samples		
NN	.0344	.015
DT	.0445	.032
STAT	.0948	.018

Table 5.1: **Burglar Alarm Database Mean Errors**

This shows the mean errors achieved by STAT, DT, and NN on the burglar alarm database.

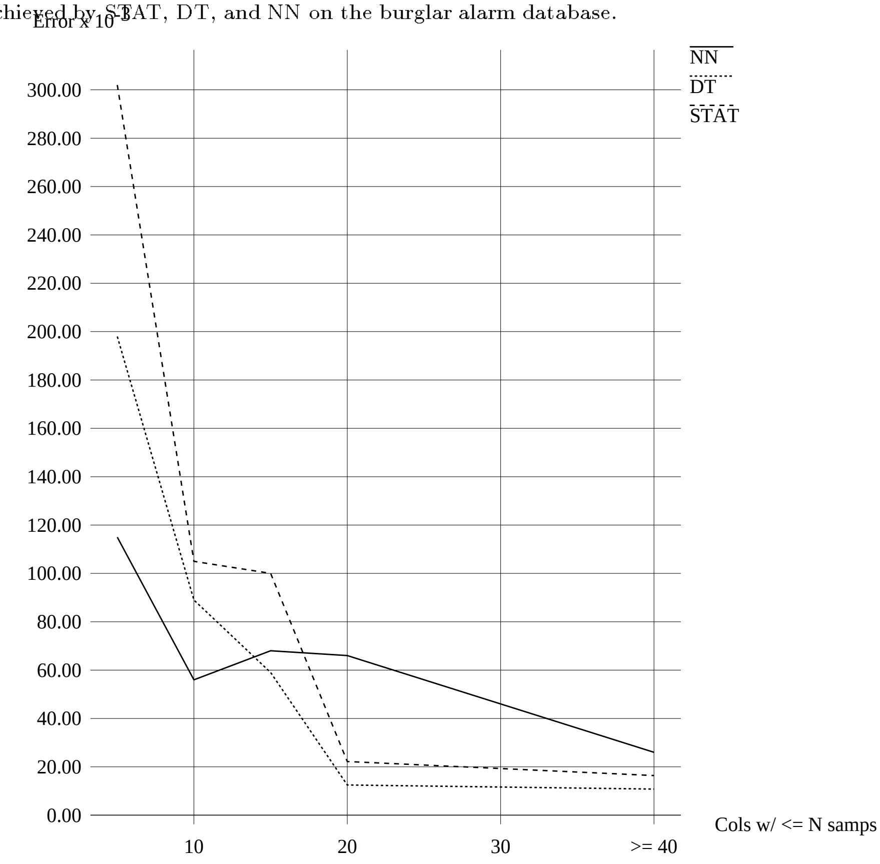


Figure 5.3: **Burglar Alarm Database Results, 100 samples**

STAT, NN and DT on the alarm database for 100 samples. The NN was trained for 20 epochs.

trend is for high error on the data-poor columns to the left, and low error on the data-rich columns to the right. This makes perfect sense. Consider STAT. The more data that STAT sees for one column, the more accurate its assessment of that column will be.

For the 100-sample graph, there are a couple of strange glitches in the data around the 15

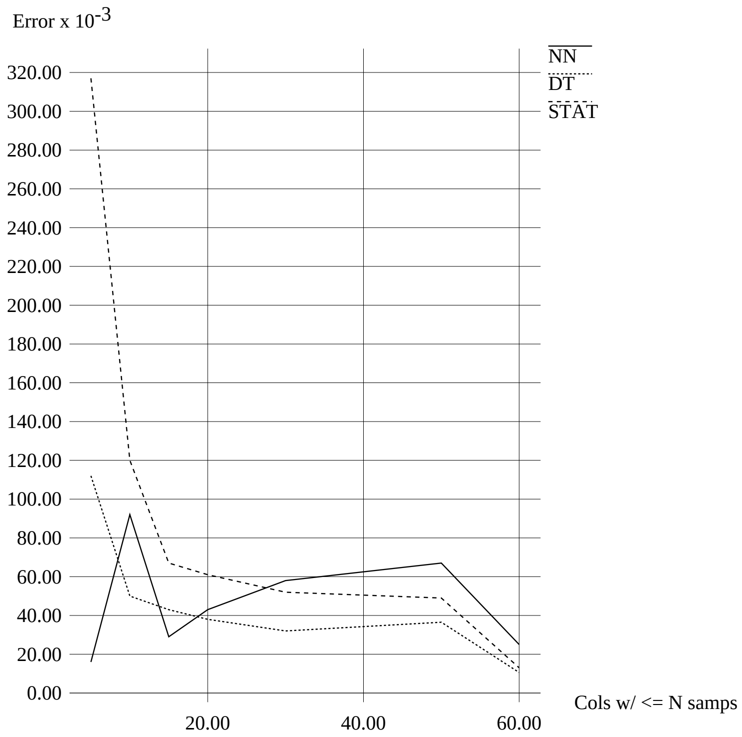


Figure 5.4: **Burglar Alarm Database Results, 500 samples**
STAT, NN and DT on the alarm database for 500 samples. The NN was trained for 50 epochs.

and 20 sample points. This is because there were actually only 2 and 1 data point(s) respectively for these cases. The 500 sample graph has the same problem at the 10 sample point. This is also attributable to the small size of this problem. The car insurance database in Section 5.4.3 has a large number of cases per bucket, giving a clearer picture in general of the error curves.

There are several basic trends to notice here. The most obvious is that DT is consistently performing better than STAT, at all points on both curves.

Second, for data-rich columns, all techniques perform very well. We would expect this for STAT, as we know from Chapter 3 that the variance of the beta distributions that STAT uses to learn the CPTs is a linearly decreasing function of the number of relevant samples. It is comforting to see both NN and DT approaching or outperforming STAT on these data-rich columns. This indicates the correctness of the implementation and the overall accuracy of the approaches.

Third, on the data-poor columns, NN and DT performed consistently better than STAT. In general, STAT doesn't seem to surpass NN in accuracy until columns with about 20-30 samples are reached.

Finally, for very data-poor columns, NN is by far the best of the three methods, and has been consistently throughout these trials.

Figure 5.5 (the last for this database) shows the learning behavior as the total number of samples ranges from 10 to 500. The neural network was run for 30 epochs on this data set.

This graph makes a very important point. NN (and DT) is achieving higher accuracy on

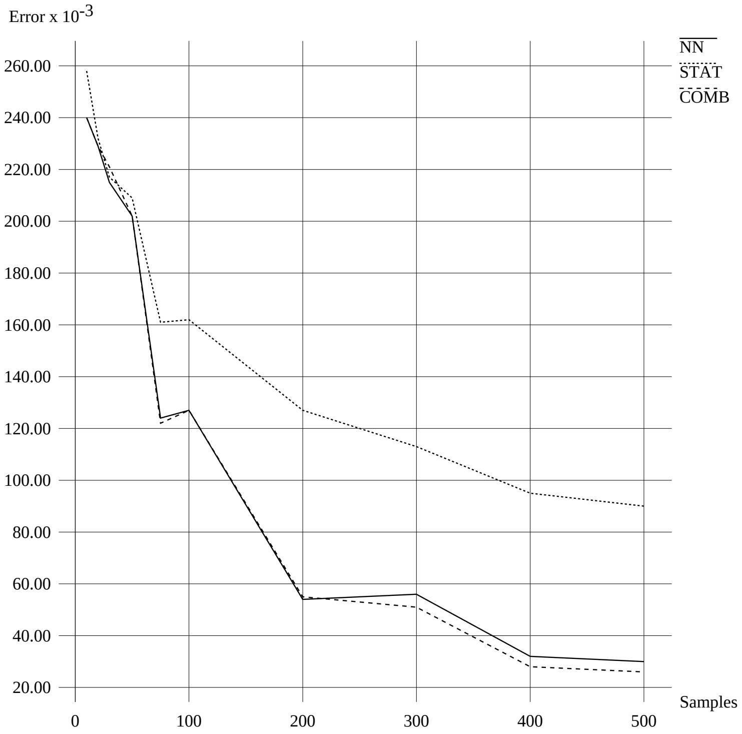


Figure 5.5: **Burglar Alarm Database, Increasing Sample Size**
NN STAT and COMB on the alarm database. NN was trained for 30 epochs. This shows the relative gains in accuracy for both techniques as the overall number of samples is increased.

data-poor columns by using information from data-rich columns. When that information is not there, then the performance is similar to that of STAT at about 20 to 25% error rates. This is especially the case with the data-poor columns – on these columns mean errors were commonly on the order of .3 or .4. There seems to be a big gain in NN accuracy at about 50. This is mainly because there are several low probability columns in the database, and at this point those columns start to see a couple of examples. NN takes much greater advantage of this information than STAT. Finally, note that as the sample size increases, COMB is outperforming both NN and STAT.

Approach	50 Samp	100 Samp	500 Samp
NLR	.081	.058	.030
DT	.084	.061	.029
NN	.088	.071	.041
STAT	.094	.070	.034
COMB	.087	.062	.035

Table 5.2: **Dog-Out Database Mean Errors**

This shows the mean errors on the dog-out database for all five approaches tested. NN was run for 30 epochs.

5.4.2 Dog-Out Database

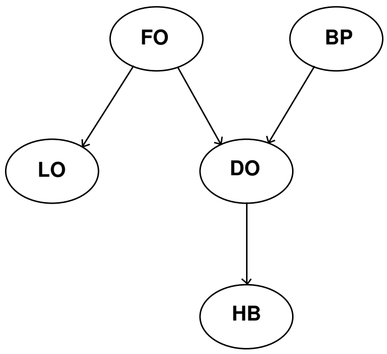


Figure 5.6: **Dog-Out Database**

Repeated from Chapter 3. All variables are binary. FO = fo when the family is out, **fo** when not. Similarly, BP = bp when the dog has bowel problems, LO = lo when the outside lights are on, DO = do when the dog is out, and HB = hb when you can hear the dog barking.

The belief net and database are the same as described in Chapter 4. For convenience, the graphical structure is repeated here in Figure 5.6. The results for learning this problem are shown in Figure 5.7 and Table 5.2.

There are a couple of points that are readily noticed. Looking at the DT curve, it is clear that the DT approach is prone to wild swings in accuracy compared to either NN or STAT. But, from the table it is also clear that again, it has an excellent mean error performance. The lesson to take from that seems to be that DT is making fairly large inductive leaps that more often than not correct. But when it is wrong, it tends to miss by quite a bit.

Also, notice that it is very difficult to predict where DT will outperform STAT. From the graphs we have shown, it appears that DT is consistently outperforming STAT on the data-rich columns, but this is not the case with other runs that we have not presented, where DT performed worse by about 3-4%.

NN, however, for the most part is better than STAT on the data-poor columns. It is very interesting to note that on the 50-sample and 100-sample cases, the mean error for NN showed a modest 6.4% gain and a -1.4% loss over the STAT mean errors, but COMB produced 7.4%

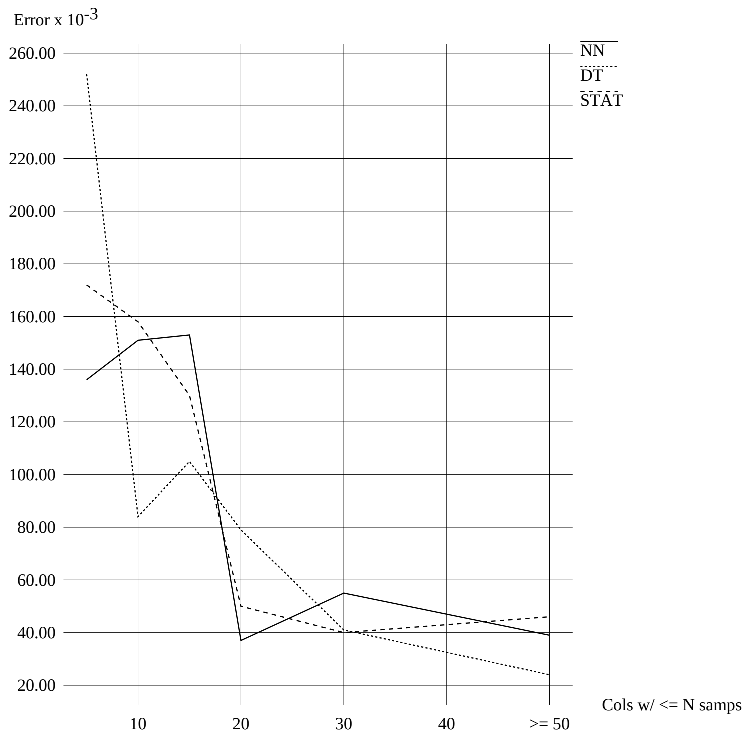


Figure 5.7: **Dog-Out Database, Results**
This shows the 50 sample case for DT, NN, STAT. NN was run with 30 epochs.

and 11.4% gains over STAT. The datum in Table 5.2 that is the most interesting is the 100-sample COMB point. Even though STAT and NN mean errors are both hovering around .07, the combination of the two produces a mean error of about .06. On the 500-sample case there were few data-poor columns and thus STAT shows strong results.

The final comment we make in this section is in regards to NLR, the nonlinear regression method. The problem again is that PIMPLE (our NLR technique) expects predictors with several different values, implying that it can not be applied to a binary node, or to a node with binary parents. In order to test our ideas, what we did was to add some random white noise to every value of our binary database, with a signal-to-noise ratio of 15:1. The result of this is shown in the table. PIMPLE generated predictions that were about 15% better than STAT, even though STAT was running off the clean data.

5.4.3 Car Insurance Database

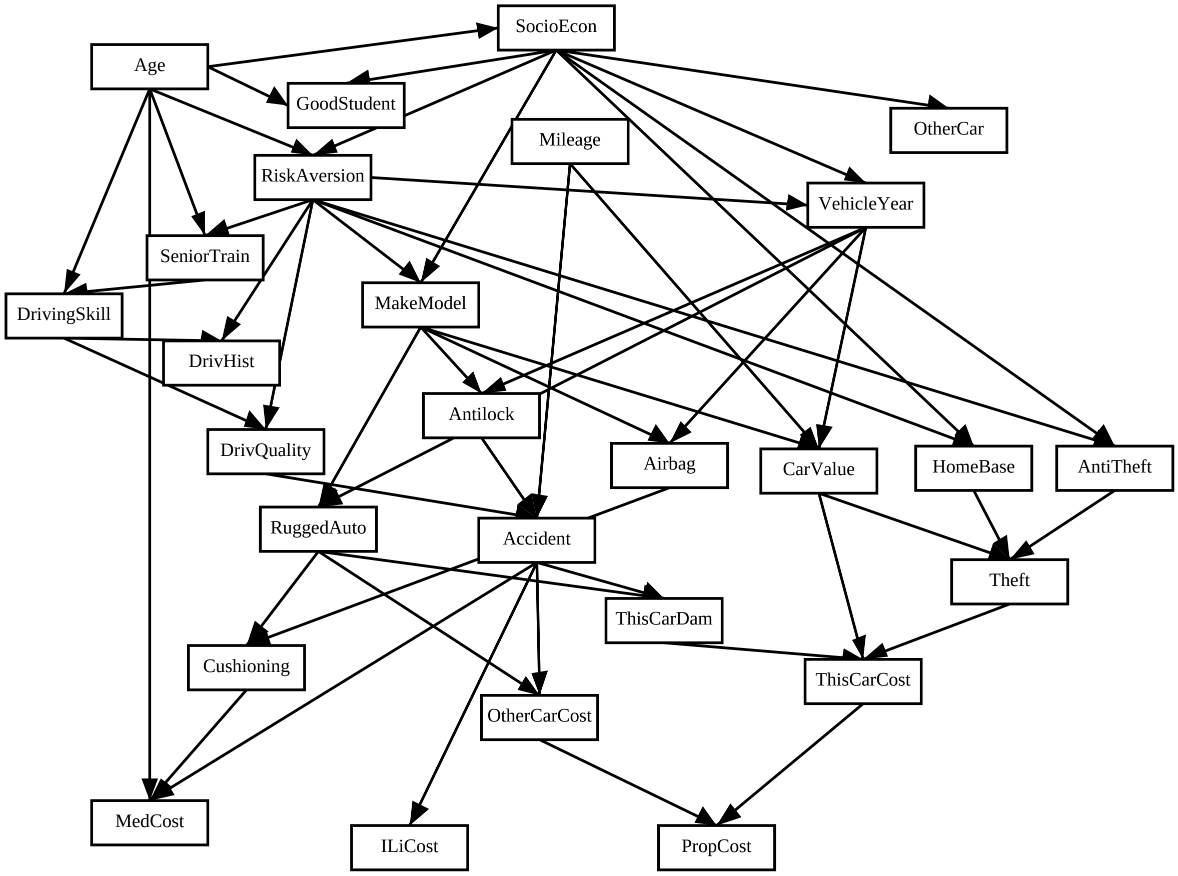


Figure 5.8: **Car Insurance Database**

Repeated from Chapter 3, this is the model of an insurance database, it includes binary, nominal, discrete and continuous variables.

This is our most significant network (courtesy of Stuart Russell), with 27 binary, discrete, nominal, and continuous nodes, 52 arcs and over 1400 conditional probabilities. It is the car insurance database shown in Figure 5.8, where the goal is to help predict how much financial risk is run from various policy holders, given data about age, good student, socio-economic class, and so on. Keep in mind that in high volume domains like this, a 1% improvement in risk assessment could be worth millions or even billions of dollars.

The difficult to understand variables in the network are: *MedCost* (medical cost), *PropCost* (property cost), *IliCost* (intangible liability cost), and *SeniorTrain* (senior training, such as drivers ed. for seniors). The rest of the variables are self-explanatory.

Figure 5.9 shows the learning results for this database on 100 samples, running the neural net for 30 epochs. Note the excellent performance of NN in this complex case, well outperforming STAT over most points, and doing essentially the same for columns with more than 50 samples.

Figures 5.10 and 5.11 show the performance of the learning as the total number of samples

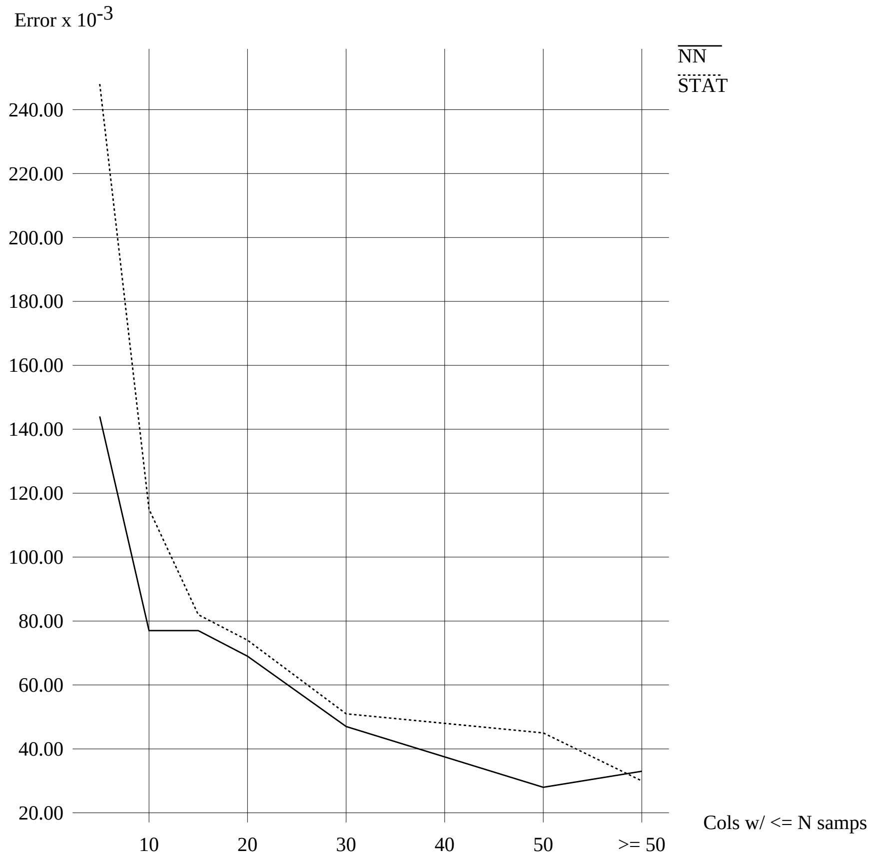


Figure 5.9: **Car Insurance Database Results**
The results of applying NN and STAT to the car insurance database on 100 samples, 30 epochs.

is varied from 50 to 1000. The learning curves in Figure 5.10 were built using the mean error metric, the curves in Figure 5.11 using the weighted probability metric as discussed in beginning of Section 5.4.

The comparison of the two figures provides some interesting insights. Figure 5.11 shows the STAT curve outperforming the NN curve at about 350 samples and more. This is because the high probability columns are the data-rich columns, and the data-rich columns are counted the most heavily in the probability-weighted metric. Since we expect STAT to do well estimating data-rich columns, it is no surprise to see the NN and STAT curves cross. What is interesting is to see that COMB, under both metrics, is significantly outperforming NN and STAT. Even with the probability-weighted metric on the largest sample (1000 instances), the contribution from NN on the data-poor columns in COMB leads to a 10% improvement over STAT.

It seems evident from Figure 5.11 that if we continue increasing the number of samples supplied to the system, STAT will eventually converge to COMB. This is again as we would expect.

5.4.4 Summary

The DT code we used in this testing is fairly simple in terms of implementation and ability. Since it only handles binary variables, we were not able to test it on the car insurance database. It is fast. Where as NN was taking an hour to complete ten sets of 20-epoch 100-sample runs on a

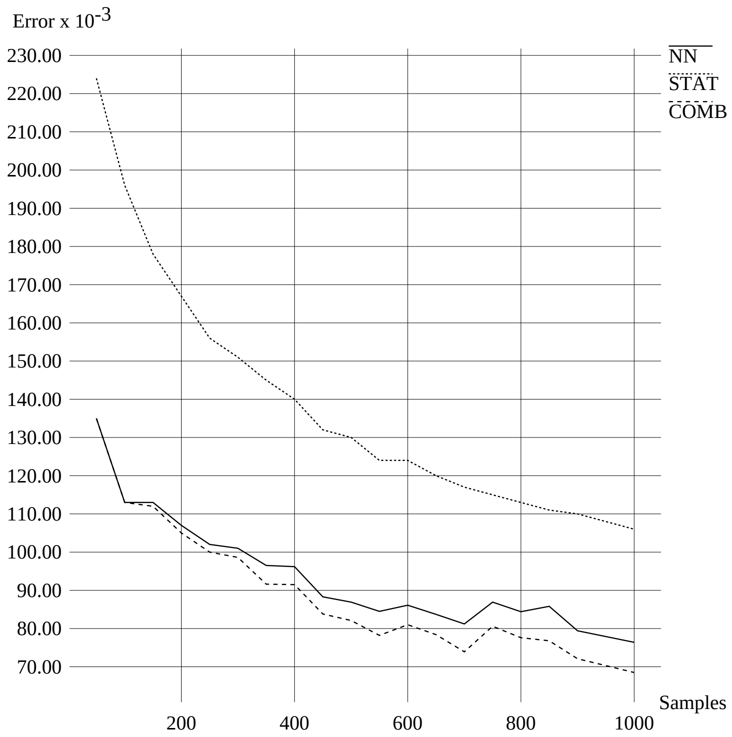


Figure 5.10: **Car Insurance Database, Mean Error Metric**

The results of applying NN and STAT to the car insurance database as the sample size is increased to 1000 samples. This graph uses the mean error metric.

Dec5100, the decision tree code was completing in just a few minutes. But the results were mixed in that on some problems the performance is stellar, substantially outperforming STAT at every point. In other cases, it could barely keep up.

NN worked very well in general. One major criticism of the implementation is that the data-poor columns tended to get pushed towards 0 or 1 in many cases. It seems that what is happening is largely due to the momentum term. Whichever way the weights change first (equivalently whichever way the output unit is pushed first) will be the way that it keeps moving, in general. This is because the data is often not available to overcome that initial push. In general though this problem seems to be fairly limited, and the results speak for themselves.

A whole slew of other (similar) databases were used in testing these ideas, but it is not useful to include them here. The main point to be made in this section is that the experimental results have supported the thesis, that the results from bookkeeping techniques can be significantly improved. We have shown that both the DT and the NN implementations commonly achieved mean errors that were 25% to 60% better than STAT, and NN consistently outperformed STAT in predicting data-poor columns. We have shown that a combination of STAT and NN proves even more fruitful in terms of mean error gains, and has the additional benefit of leaving the data-rich columns with beta distributions to be used in inference.

Another possible advantage of improving the learning of low-probability statements is that

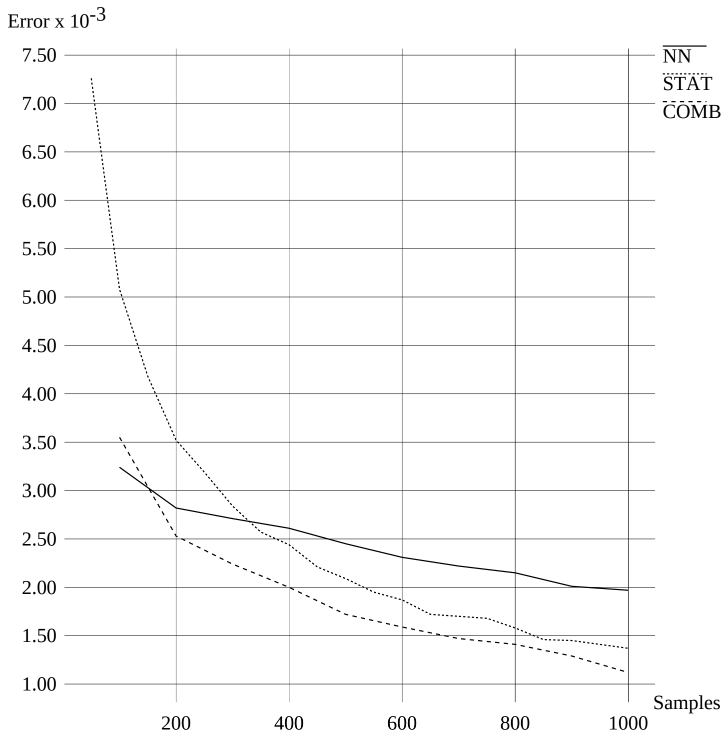


Figure 5.11: **Car Insurance Database, Weighted Probability Metric**

The results of applying NN and STAT to the car insurance database as the sample size is increased to 1000 samples. This graph uses the probabilistic weighting metric.

it allows us to add extra parents to nodes. Adding parents to the nodes complicates the learning task for that node, multiplies the amount of data needed to cover the node by the number of instantiations of the new variable, but adds another degree of discrimination that might be very useful. If the data exists and can be taken advantage of appropriately, then adding this extra predictor will be worth it in some cases.

5.5 Learning Hidden Nodes

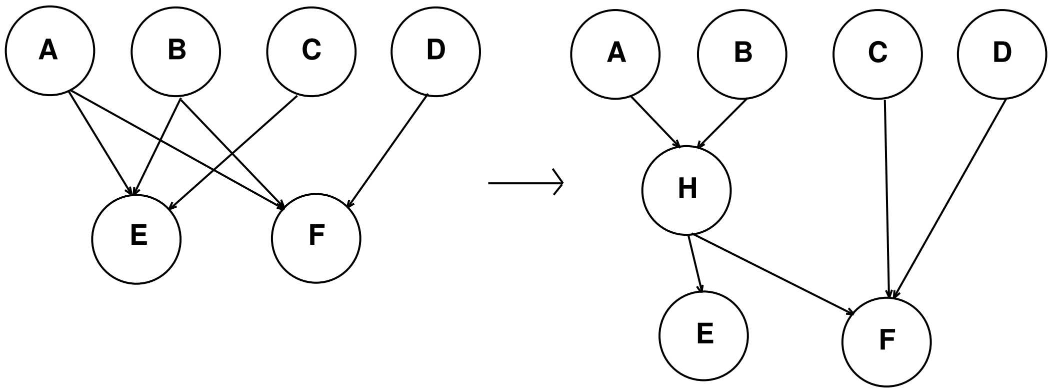


Figure 5.12: **Addition of a Hidden Node**
The hidden node H is added to the network, resulting in significant reduction of the space required to represent the CPTs.

A very exciting idea in belief network induction is the introduction of hidden nodes to simplify the learning problem. The idea is that if we can take a cluster of related nodes in a network, and add a new node in such a way as to reduce the total number of conditional probabilities in that cluster, then that new network will have fewer parameters to learn, and thus we should be able to learn it better. Another benefit is that this node helps to explain the causal processes that exist in the domain in the same way that a typical node in a belief network does.

Consider the following example in Figure 5.12. Assume that each node has 5 values. Then the belief net on the left has a total of 1270 conditional probabilities, whereas the belief net on the right has only 395. With larger networks the reductions can be even more dramatic.

The basic problem with adding hidden nodes is that *there is no data that describes H*, our hidden node. How should we fill out the values of the CPTs for the nodes E and F above if we have no information on H? There is some information that can help in determining these values, this information consists of a set of constraints that come from the CPTs for E and F in the original belief network. Note that this problem is identical to having a pre-existing *unmeasurable* variable in a network.

The majority of this section concerns a method that trys to exploit those constraints in order to come up with a set of values for H and the new E and F. We have met with only limited success with this approach, and have run into some seemingly very hard and thick walls. We explain the approach below, and examine its limits, showing that this method probably can not be extended to handle more complicated hidden node problems. Then, in Section 5.5.3, we talk about an alternative to learning hidden nodes that does not provide knowledge of the causal processes in the belief network, but does significantly reduce the number of parameters being learned.

5.5.1 The simplest case

We will use a simpler cluster of nodes to explore the approach taken, depicted in Figures 5.13, and 5.14.

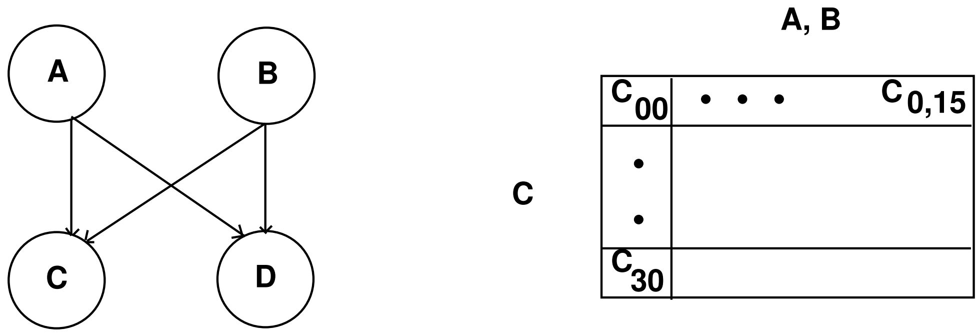


Figure 5.13: **Network without Hidden Node**

The original structure shown before the addition of the hidden node. The CPT for node C is shown as well; these values are all known. All nodes have 4 values.

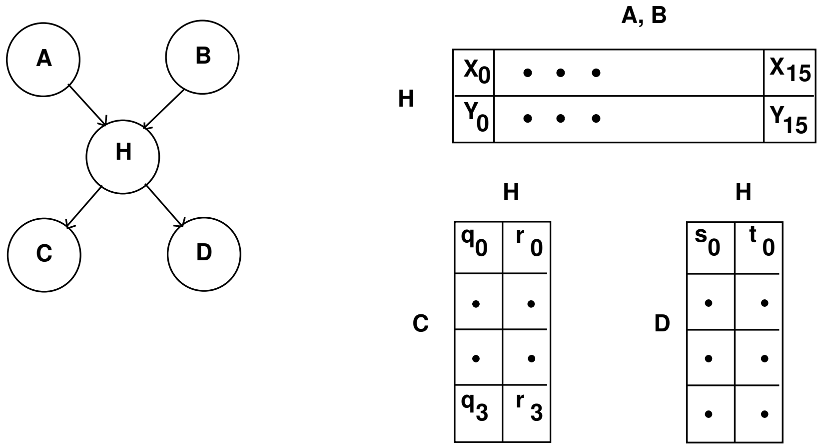


Figure 5.14: **Network with Hidden Node**

The structure after the addition of the hidden node. The CPTs for nodes C, D, and H are shown as well; these are unknown. A, B, C, and D all have 4 values, and H is binary.

All of the variables except H have 4 values; H has 2. For the network without the hidden node, there are a total of 102 independent conditional probabilities to be learned whereas the network with the hidden node has 34 independent probabilities. The idea is that we start with the network shown in Figure 5.13 and learn the CPTs. These values will be used to constrain the CPTs in the network with the hidden node. Then, insert a binary hidden node as in Figure 5.14, and determine the CPTs for the new C, D and H nodes from the information from the network without the hidden node. How can this be determined? We know that

$$Pr(C|A,B) = \sum_H Pr(C|H)Pr(H|A,B)$$
$$Pr(D|A,B) = \sum_H Pr(D|H)Pr(H|A,B). \tag{5.5}$$

That implies that if we can solve the equations

$$\begin{aligned} C_{ij} &= q_i x_j + r_i y_j \\ D_{ij} &= s_i x_j + t_i y_j \end{aligned} \tag{5.6}$$

for the unknowns q_i, r_i, s_i, t_i, x_j and y_j, then we have solved for the hidden node. In fact, if the original network was a MAP (maximum aposteriori probability) assignment of probabilities given the structure and the data, then this transformation to the network with the hidden node will also be a MAP assignment of probabilities. The bookkeeping (STAT) procedure is fairly easily proven to be the MAP assignment of probabilities, given the equations derived by Cooper and Herskovits [22].

The rest of this section outlines the method of solving Equations 5.6. The results of this approach gave 10-15% gains on the mean errors of the simple tables learned.

The equations in 5.6 are nonlinear, and in the current form they are rather difficult to solve directly. The nonlinearity can be removed after some extensive algebraic manipulation, ending up with equations of the form:

$$\begin{aligned} r_1 &= a_1 + b_1 r_0 \\ r_2 &= a_2 + b_2 r_0 \\ r_3 &= a_3 + b_3 r_0, \end{aligned} \tag{5.7}$$

where the a_i and b_i terms are constants composed of the C_{ij} and D_{ij} probabilities from the CPTs in the open networks. Similar equations can be written for q_i, s_i and t_i as well. These equations do not in general have unique solutions. This is problematic since we need to make sure that the solution picked for node C (r_i and q_i) "corresponds" to the solution picked for node D (s_i and t_i). To visualize this better, imagine two intersecting planes (representing all possible solutions for C and D), and the line of their intersection (which represents the fact that in the hidden node network the nodes C and D are linked together by H, and the probabilities in H are constrained to be between 0 and 1, and to sum to 1). We need solutions that fall on this line.

This additional constraint is derived from relationship between the hidden node H and the two tables $Pr(C|H)$ and $Pr(D|H)$. From this relationship we can produce equations relating the quantities q_i, s_i and t_i to r_0, and in doing so *in theory* we provide enough constraints to allow a complete solution. After solving for r_i, q_i, s_i and t_i, we go back to the equations in 5.6, plug in the values, and solve the now *linear* equations for the CPT for H.

The problem with the current analysis as it stands is that we don't have the actual values for C_{ij}, D_{ij}, just their approximations. The approximations are not good enough, so the linear equations commonly have no solution.

What we do then is to solve for the minimum squared error MAP probabilities for the new network. This is done by solving the same equations for r_i detailed above, with a twist. Using the simplex method, we have a set of linear constraints on r_i, and we add an objective function to be minimized. Simplex then finds the r_is that satisfy the linear constraints, and minimize

$$\sum_i (r_i - a_i - b_i r_0)^2. \tag{5.8}$$

This minimization term comes directly from Equations 5.7. The Simplex method normally requires linear objective functions, which is not consistent with Equation 5.8. There is a trick that can be applied here to overcome this problem, it is called the cutting plane method. What this does is to approximate the nonlinear objective function with a set of linear tangents. After each tangent is added, the normal Simplex is run, and the result of that run is used to compute the new tangent. This iterative approach converges to the actual solution in the long run.

This approach was implemented, the result being a 10-15% improvement in the mean error of the tables for C and D.

5.5.2 Problems with extension

Problems arose when extending the ideas to more complex cases, for example the structure in Figure 5.12, where the children E and F have a mixture of parents including the hidden node H. There were several significant obstacles to overcome in making this extension, but they are not described here as they are irrelevant to the end result. The results were very disappointing, with the hidden node network giving on average 30% higher mean error rate on the children nodes E and F.

There are two major difficulties in extending this "equational" approach to determining the CPTs. Perhaps the largest is that in the extension, a very important constraint gets lost.

In the simple case, the hidden node is the only parent of both node C and D. Because of this, the x_j and y_j variables in Equations 5.6 are identical (in other words, x_1 in the top equation equals x_1 in the bottom equation). Removing x_j and y_j by solving for them is the step that links the new CPT for node C with the new CPT for D. This link is in the form of the constraints on q_i, s_i and t_i that depend on the solution of r_i. As mentioned in the section above, this link is extremely important because there are many solutions to minimizations of the form of Equation 5.8. What it does is to assure that the solution picked for C corresponds to the solution picked for D.

The extended case no longer has this constraint, because the introduction of the extraneous parents. The new form of Equation 5.6 is:

$$\begin{aligned} Pr(E|A,B,C) &= \sum_H Pr(E|H,C)Pr(H|A,B) \\ Pr(F|A,B,D) &= \sum_H Pr(F|H,D)Pr(H|A,B). \end{aligned}$$

The problem is that fixing the parents of E to a particular value, say A=0, B=0, and C=0 does not fix the parents of F to any value, since nothing has been said about the instantiation of D. This effectively cuts the link between E and F. Losing this constraint gives the Simplex method a free hand in picking any solution on two planes representing the probabilities $Pr(E|H,C)$ and $Pr(F|H,D)$, and more likely than not the these solution points do not both fall on the same point of the "line" defined by the hidden node. To simplify this argument, it is sufficient to say that we no longer have the option to use a vital constraint.

The second problem that exists may not be as critical as the above problem, but it is still serious in nature. The information we are using for this equational approach is all from sparse tables. We are basically taking several very wide underdetermined distributions, and trying to combine them in some way that makes sense. But the algebraic process of combination is sensitive to the errors in prediction (it is ill conditioned), and therefore tends to increase the uncertainty in the result.

It seems that the equational approach for complex hidden nodes (or, real life hidden nodes) simply does not work, as important constraints disappear and extended algebraic manipulations degrade the quality of the information available. It is more likely that an approach based on a gradient descent method will have a better chance.

5.5.3 Neural Nets vs. Hidden Nodes

One reason for inserting the hidden node in the first place is to reduce the number of parameters that need to be learned, thereby increasing the sample density over the new domain.

An alternative to inserting a hidden node is to instead learn the child nodes in question with a neural network. This approach will say nothing about the causal process underlying the network, but it will help reduce the number of parameters being learned. The number of weights and hidden nodes in the neural networks that we have used for CPT learning is a much smaller quantity than the number of cells or columns in a large table. Take for example Figure 5.12. There are 1270 conditional probabilities. Inserting a hidden node reduces the probabilities needed to 395. If we instead apply the neural network construction we have used in the implementation, then the learning is done over 20 conditional probabilities external to the neural network, plus 104 parameters internal to the neural net.

The reduction is not significant for nominal nodes, but tremendous for other nodes. The reason there is such a huge reduction in the number of parameters required to represent the system is due to the input space compression. Each parent node of X_i, no matter whether binary, discrete or continuous is turned into one input unit for the network; all the distinct values for that node are compressed to fall between -1 and 1. Of course, with more complex neural network constructions than those found in this dissertation the reduction may not be so dramatic. Also, the effectiveness of the neural network applied in this way will ultimately depend on the types of generalizations that the neural network can make compared to the types that must be made.

Chapter 6

Conclusion

To conclude, this chapter reviews the contributions of this research program and points to current limitations and possible further work.

6.1 Contributions

The area of database mining is very young, but the basic task of making more sense out of the data at hand has been one that humans have faced since the beginnings of time. Only recently has the problem grown from being one of making good decisions based on limited data to one of making sense of huge amounts of data. The driving force behind this change has been the advent of database technology which has lead to huge collections of data that will only get larger as time goes on. Database mining is the science of how to manage this flood of information. The goal of this dissertation has been to provide a coherent and effective approach to database mining based on probabilistic modeling and reasoning technologies.

Specific areas of contribution are

- **Inference distributions in belief networks**. Any system that induces models from data, or constructs models from domain experts *must* be able to attach a measure of belief to any answer that the model produces. Without a measure of how much data is supporting a result it is difficult to have any faith in the answers. We have shown theorems demonstrating how to maintain *beta distributions* when computing inferences in the network. These results are applicable any time the CPTs store beta distributions, which will always be the case when doing statistical induction. When other learning approaches are used, beta distributions for the CPTs can be estimated.
- **A decision theoretic sampling theory for scalability**. There are two related issues in scaling database mining techniques to real problems. One is that databases are large and growing larger, so these approaches must be able to handle enormous amounts of data. The second issue is that we exist in a time-critical domain. We often do not want answers that are as accurate as possible. Instead we need answers of specifiable qualities as fast as they can be churned out. BNI has shown a *decision theoretic sampling theory* that calculates the likely sample size required to induce models that are "good enough" for current user

purposes. These ideas are not limited to belief network induction, in fact they were originally demonstrated for the problem of constructing decision trees.

- **New approaches to belief network induction**. Statistical induction is the standard approach for updating beliefs both in databases and belief networks alike. This requires too much data to get reasonable results. BNI instead *integrates other learning approaches* such as neural networks, nonlinear regression and decision tree algorithms into the belief network induction task, and shows results that demonstrate dramatic improvements in model accuracy.

- **Divide and conquer approach to modeling**. Learning a belief network model tends to reformulate the learning task into many small, localized CPT learning subtasks. The *significant reduction in complexity* of these smaller learning problems leads to an overall approach that needs much less space to run, is faster, and will often be more accurate than a straightforward application of (for example) a neural net to the modeling problem. There is even more potential for speedup since the CPTs can be learned independently of each other, implying that the approach is highly parallelizable.

- **Implementation that demonstrates goals**. The theories put forth in this dissertation have been implemented, and these implementations have demonstrated the feasibility and effectiveness of the ideas.

6.2 Limitations and Further Work

This research project has pointed out several potential areas for further exploration and research.

- **Inference**. In order to get the full benefit from the theorems that show how to maintain the beta distributions, two additional problems must be solved. First, the inference approach based on the four transformations node splitting, node merging, arc reversal and node removal must be mapped into a speedier exact inference technique like the junction tree approach. Second, methods for approximating beta distributions need to be explored for the other learning techniques (neural networks, decision trees, regression) that are attached to the belief network. Solutions to these two problems will permit relatively fast production of inference distributions no matter which technique has been used to learn the CPTs.

- **Sequential sampling**. More work needs to be done on the sampling theory in order to step away from the one-shot sampling approach to move to a theoretically justifiable *sequence* of samples. We have provided a push in this direction, but more must be done in order to remove the ad-hoc flavor of the solution we provided.

- **Utility theory**. The concept of utility is central to inducing "interesting" models of the data, producing answers that are "good enough", extracting new interesting results from data, and evaluating the usefulness of the learned models. We have introduced and explained the need for utility theory in several places in the dissertation, but have not gone beyond applying the simplest concepts of utility theory to any of the problems examined. None of these questions can truly be answered in a normative sense without the concept of utility theory firmly enmeshed in the results.

- **Hidden nodes**. The problem of learning with hidden nodes is one of the most difficult problems facing belief network induction. The need for solving the problem is apparent, for there will always be domains with variables that are not directly measurable. To date, there is still no completely satisfactory answer, nor even an approach that seems to have the potential to solve it adequately.

- **Ethics**. Great care must be taken in restricting the types of information that can be learned and output with these techniques; the potential for invasion of the privacy of the individual is far too great to be ignored. The ethical boundaries of database mining need to be established and enforced internally to avoid the public backlash that will occur if these techniques are abused.

The difficulties left to the fields of database mining and machine learning are significant and will grow ever more demanding as our databases expand. In solving these problems we can not help but to further our understanding of the methodologies behind problem solving, decision making, and rational intelligence. It is this constant challenge of new and interesting problems that insures the continued advancement of machine intelligence that goes hand in hand with the unstoppable progression of human knowledge and knowhow.

May our wisdom grow with our abilities.

Appendix A

Beta Distribution Preliminary

The beta distribution is mainly used to characterize random phenomenon whose set of possible values is in some interval $[c, d]$. This dissertation deals mainly with probabilities as the random variables, thus insuring the intervals to be $[0, 1]$. A random variable θ has a beta distribution if its density function is:

$$f(\theta) = \begin{cases} \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1} & 0 < \theta < 1 \\ 0 & \text{otherwise} \end{cases}$$

where $B(a, b)$ is a constant equal to

$$\begin{aligned} B(a,b) &= \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta \\ &= \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \end{aligned} \tag{A.1}$$

Mean μ and variance σ^2 for $\theta = \beta(a, b)$ is found as:

$$\begin{aligned} \mu &= \frac{a}{a+b} \\ \sigma^2 &= \frac{ab}{(a+b)^2(a+b+1)} \end{aligned}$$

Note that the statistics a and b are sufficient to completely describe the beta distribution; these are called the *sufficient statistics* of the beta distribution.

As can be seen in Figure A.1, the shape of the beta is very similar to the normal distribution when the mean is at .5. As the mean moves towards 0 or 1, the distribution looks like a normal that has been somewhat *pushed* on near the top. Conceptually, this makes sense because the beta is restricted to be within the interval $[0, 1]$, whereas the normal extends to infinity. If the mean is near the boundary, then much of the probability mass must also be near the boundary, but none of it may extend past it.

The beta distribution is closely related to the Dirichlet distribution, in fact the Dirichlet is a n-dimensional beta distribution. The Dirichlet $D(a_1, a_2, \ldots, a_n; a_{n+1})$ has the following density function:

$$f(\theta_1, \ldots, \theta_n) = \frac{\Gamma(a_1 + \ldots + a_{n+1})}{\Gamma(a_1)\ldots\Gamma(a_{n+1})}\theta_1^{a_1-1}\ldots\theta_n^{a_n-1}(1-\theta_1-\ldots-\theta_n)^{a_{n+1}-1}.$$

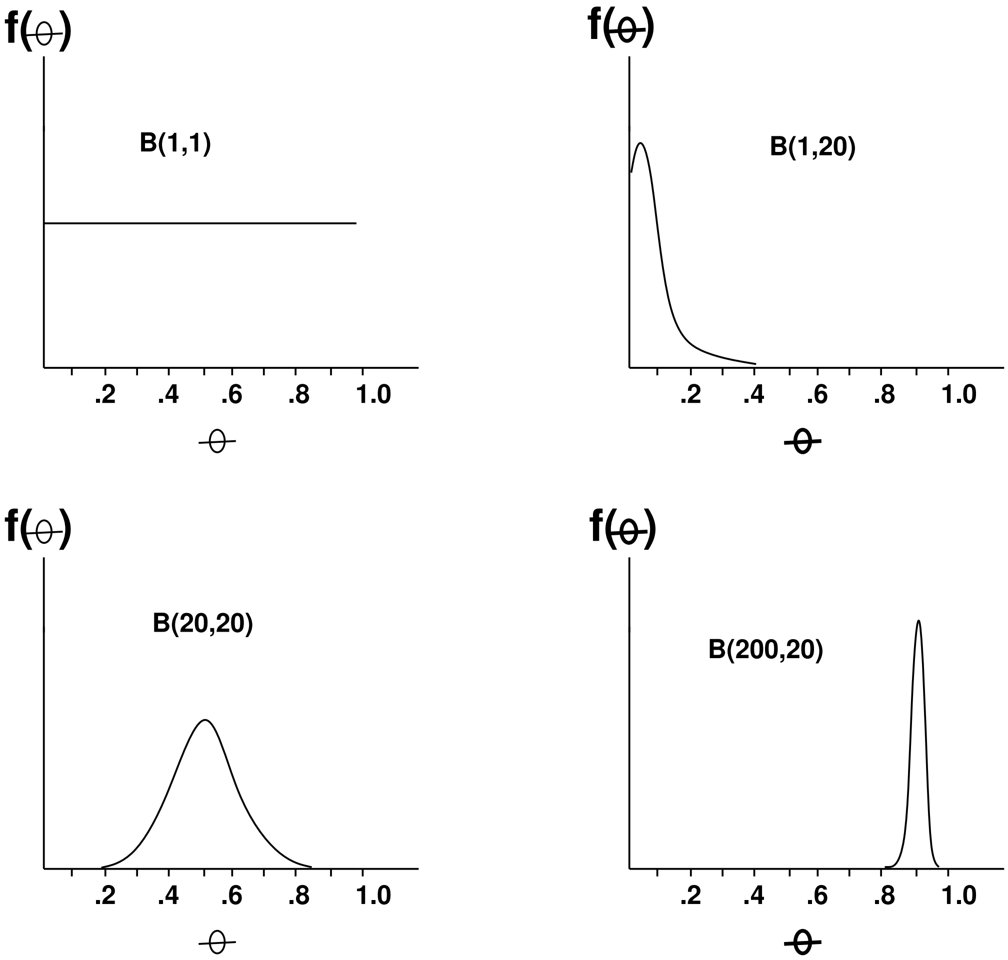


Figure A.1: **Some Beta Distributions**
Four beta distributions.

Appendix B

Conditional Probability Tables

In the following sections, “t” is used for “true” and “f” for “false”.

B.1 Burglar Alarm Database

These are the conditional probability tables for Figure 5.5.

Pr(Theft)		
Theft	t	.1
	f	.9

Pr(Quake)		
Quake	t	.01
	f	.99

Pr(Alarm | Quake, Theft)					
		Theft,Quake			
		t,t	t,f	f,t	f,f
Alarm	t	.99	.95	.8	.01
	f	.01	.05	.2	.99

Pr(J calls | Alarm)			
		Alarm	
		t	f
J Calls	t	.8	0
	f	.2	1.0

Pr(M calls | Alarm)			
		Alarm	
		t	f
M Calls	t	1.0	.2
	f	0	.8

B.2 Dog-Out Database

These tables correspond to the dog-out database, the belief network can be found in Figure 5.6.

Pr(FO)		
FO	f	.68
	t	.32

Pr(BP)		
BP	f	.88
	t	.12

Pr(LO | FO)			
		FO	
		f	t
LO	f	.10	.29
	t	.90	.71

Pr(DO | FO, BP)					
		FO,BP			
		f,f	f,t	t,f	t,t
DO	f	.80	.5	.41	.14
	t	.20	.5	.59	.86

Pr(HB | DO)			
		DO	
		f	t
HB	f	.89	.05
	t	.11	.95

B.3 Car Insurance Database

Some of these tables are rather large. The path "http://http.cs.berkeley.edu/ musick/" on the world wide web will show the hugin input file for these tables, which can be viewed in plain text via Mosaic or some similar program, or in Hugin if it is available.

Bibliography

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions.* Dover Publications, New York, 1972.

[2] R. Alterman and M. Wentworth. Determining the important features of a case. In *Proceedings: Case-Based Reasoning Workshop.* Morgan Kaufmann Publishers, 1989.

[3] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[4] R. E. Bechhofer and D. M. Goldsman. Truncation of the Bechhofer-Kieffer-Sobel sequential procedure for selecting the normal population which has the largest mean. *Communications in Statistics: Simulation and Computation*, 16(4):1067–1092, 1987.

[5] R. E. Bechhofer and D. M. Goldsman. Truncation of the Bechhofer-Kieffer-Sobel sequential procedure for selecting the normal population which has the largest mean (III): Supplementary trucation numbers and resulting performance characteristics. *Communications in Statistics: Simulation and Computation*, 18(1):63–81, 1989.

[6] R. E. Bechhofer, J. Kiefer, and M. Sobel. *Sequential Identification and Ranking Procedures.* The University of Chicago Press, Chicago, Illinois, 1968.

[7] J. S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 1992.

[8] J. S. Breese and E. J. Horvitz. Ideal reformulation of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 64–72, 1990.

[9] L. Breiman. The $\prod$ method for estimating multivariate functions from noisy data. *Technometrics*, 33(2):125–160, 1991.

[10] B. G. Buchanan and E. A. Feigenbaum. DENDRAL and Meta-DENDRAL: their applications dimension. *Artificial Intelligence*, 11:5–24, 1978.

[11] W. Buntine. Theory refinement on Bayesian networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

[12] J. Catlett. *Megainduction: machine learning on very large databases.* PhD thesis, University of Sydney, Sydney, Australia, 1991.

[13] J. Catlett. Peepholing: choosing attributes efficiently for megainduction. In *Proceedings of the Ninth International Conference in Machine Learning*, pages 49–54, 1992.

[14] K. C. Chang and R. Fung. Refinement and coarsening of Bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 475–482, 1990.

[15] E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[16] R. M. Chavez and G. F. Cooper. An empirical evaluation of a randomized algorithm for probabilistic inference. In M. Henrion, R. D. Shacter, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5.* Elsevier Science Publishers B. V., North-Holland, 1990.

[17] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: A Bayesian classification system. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[18] E. F. Codd. Relational database: A practical foundation for productivity. *Communications of the ACM*, 25(2), 1982.

[19] E. F. Codd. A relational model of data for large shared databanks. In M. Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[20] E. F. Codd. Is your DBMS really relational? *Computer World*, October 14, 1985.

[21] G. F. Cooper. A method for learning belief networks that contain hidden variables. Technical Report SMI-93-04, University of Pittsburgh, 1993.

[22] G. F. Cooper and E. Herskovits. A Bayesian method for constructing Bayesian belief networks from databases. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 86–94, 1991.

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

[24] P. Dagum, A. Galper, and E. Horvitz. Dynamic network models for forecasting. In *Proceedings of the Eigth Conference on Uncertainty in Artificial Intelligence*, pages 41–48, 1992.

[25] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, March 1993.

[26] M. H. DeGroot. *Probability and Statistics*. Addison-Wesley Publishing Company, Inc., Menlo Park, CA, second edition, 1986.

[27] G. DeJong. An approach to learning from observation. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[28] T. G. Dietterich. Learning and generalization of characteristic descriptions: An overview. In *Proceedings of the Sixth International Joint Conference on AI*, 1982.

[29] R. Duda, P. Hart, and N. Nilsson. Subjective Bayesian methods for rule based inference systems. In B. Webber and N. Nilsson, editors, *Readings in AI*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1981.

[30] D. Edwards. Extended-paulson sequential selection. *The Annals of Statistics*, 15(1):449–455, 1987.

[31] B. Falkenhainer, K. Forbus, and D. Gentner. The structure-mapping engine. Technical Report UIUCDCS-R-86-1275, University of Illinois at Urbana-Champaign, 1986.

[32] W. Feller. *An Introduction to Probability Theory and Its Applications*. Wiley, New York, 1968.

[33] K. W. Fertig and J. S. Breese. Interval influence diagrams. In M. Henrion, R. D. Shachter, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*. Elsevier Science Publishers B. V., North-Holland, 1990.

[34] D. Fisher and P. Langley. The structure and formation of natural categories. Technical Report RIA-90-02-15-1, NASA Ames Research Center, 1990.

[35] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[36] W. J. Frawley, G. Piatetsky-Shapiro, and C. J Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*. The AAAI Press, Menlo Park, CA, 1991.

[37] J. H. Friedman. Multivariate adaptive regression splines. Technical Report 102, Stanford University, Department of Statistics, 1988.

[38] G. M. Furnival and R. W. Wilson Jr. Regressions by Leaps and bounds. *Technometrics*, 16(4):499–511, 1974.

[39] A. E. Gelfand and A. F. M Smith. Sampling based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990.

[40] D. Heckerman, E. Horvitz, and B. Middleton. An approximate nonmyopic computation for value of information. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 135–141, 1991.

[41] M. Henrion. An introduction to algorithms for inference in belief nets. In M. Henrion, R. D. Shachter, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*. Elsevier Science Publishers B. V., North-Holland, 1990.

[42] M. Henrion, J. S. Breese, and E. J. Horvitz. Decision analysis and expert systems. *AI Magazine*, 12(4):64–91, 1991.

[43] E. Herskovits and G. F. Cooper. Kutato: An entropy-driven system for construction of probabilistic expert systems from databases. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 54–62, 1990.

[44] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Menlo Park, California, 1991.

[45] F. V. Jensen, K. G. Olesen, and S. K. Andersen. An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659, 1990.

[46] S. C. Kao and T. L. Lai. Sequential selection procedures based on confidence sequences for normal populations. *Communications in Statistics, Theory and Methodology*, 9(16):1657–1676, 1980.

[47] G. D. Klieter. Bayesian diagnosis in expert systems. *Artificial Intelligence*, 54:1–34, 1992.

[48] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read. Real time knowledge based systems. *AI Magazine*, 9(1):27–45, 1988.

[49] P. Langley, H. A. Simon, and G. L. Bradshaw. Hueristics for empirical discovery. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[50] P. Langley, J. M. Zytkow, H. A. Simon, and G. L. Bradshaw. The search for regularity. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 2. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[51] S. L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 1992.

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

[53] S. L. Lauritzen, B. Thiesson, and D. J. Spiegelhalter. Diagnostic systems created by model selection methods – a case study. In *Fourth International Workshop on Artificial Intelligence and Statistics*, pages 93–105, 1992.

[54] D. Lenat. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*. PhD thesis, Stanford University, Stanford, CA, 1976.

[55] D. Lenat. EURISKO: A program that learns new heuristics and domain concepts. *Artificial Intelligence*, 21(1–2):61–98, 1983.

[56] D. Lenat. The ubiquity of discovery. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[57] L. Lewinson. Data mining: Tapping into the mother lode. *Database Programming & Design*, 7(2), 1994.

[58] W. Liu. A new sequential procedure with elimination for selecting the best of several normal populations. *Communications in Statistics: Theory and Methods*, 23(11):3157–3170, 1994.

[59] D. Madigan, A. E. Raferty, J. C. York, J. M. Bradshaw, and R. G. Almond. Strategies for graphical model selection. In *Proceedings of the Eigth Conference on Uncertainty in Artificial Intelligence*, pages 331–336, 1992.

[60] C. Matheus, G. Piatetsky-Shapiro, and D. McNeil. An application of KEFIR to the analysis of health care information. In *AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 441–452, 1994.

[61] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 1983.

[62] T. M. Mitchell and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 1983.

[63] R Musick. Maintaining inference distributions in belief nets. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, 1993.

[64] R. Musick, J. Catlett, and S. Russell. An efficient method for constructing approximate decision trees for large databases. In *Proceedings of the Tenth International Conference in Machine Learning*, 1993.

[65] R. Musick and S. Russell. How long will it take? In *Proceedings of the Tenth National Conference on AI*, 1992.

[66] R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1991.

[67] R. E. Neapolitan and J. R. Kenevan. Investigation of variances in belief networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 232–241, 1991.

[68] B. E. Neuenschwander and B. D. Flurry. Principal components and model selection. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 3–14, 1993.

[69] K. G. Olesen. Causal probabilistic networks with both discrete and continuous variables. Technical Report R91-29, Aalborg Universitetscenter, 1991.

[70] S. M. Olmstead. *On Representing and Solving Decision Problems*. PhD thesis, Stanford University, Stanford, CA, 1983.

[71] G. Paab. Second order probabilities for uncertain and conflicting evidence. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 483–490, 1990.

[72] M. J. Pazzani. Indexing strategies for goal specific retrieval of cases. In *Proceedings: Case-Based Reasoning Workshop*. Morgan Kaufmann Publishers, 1989.

[73] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[74] S. K. Perng. A comparison of the asymptotic expected sample sizes of two sequential procedures for ranking problem. *The Annals of Mathematical Statistics*, 40(6):2198–2202, 1969.

[75] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*. The AAAI Press, Menlo Park, CA, 1991.

[76] G. Piatetsky-Shapiro and C. Matheus. Measuring dependencies in large databases. In *AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 162–173, 1993.

[77] G. Piatetsky-Shapiro and C. Matheus. The interestingness of deviations. In *AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 25–36, 1994.

[78] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[79] J. R. Quinlan, P. J. Compton, K. A. Horn, and L. Lazarus. Inductive knowledge acquisition: A case study. In *Applications of Expert Systems*. Turing Institute Press with Addison Wesley, Glasgow, 1987.

[80] H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. Division of Research, Harvard Business School, Boston, MA, 1961.

[81] S. J. Russell. *Analogical and Inductive Reasoning*. PhD thesis, Stanford University, Stanford, CA, 1986.

[82] S. J. Russell and E.H. Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, MA, 1991.

[83] A. Samuel. Some studies in machine learning using the game of checkers. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, 1963.

[84] H. Scheffe. *The Analysis of Variance*. John Wiley and Sons, New York, 1959.

[85] G. A. F. Seber. *Linear Regression Analysis*. John Wiley and Sons, New York, 1977.

[86] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

[87] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R. D. Shachter, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*. Elsevier Science Publishers B. V., North-Holland, 1990.

[88] S. E. Shimony and E. Charniak. A new algorithm for finding MAP assignments to belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 98–103, 1990.

[89] M. Shwe and G. Cooper. An empirical analysis of likelihood-weighting simulation on a large multiply connected belief network. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 498–508, 1990.

[90] A. Silberschatz, M. Stonebraker, J. Ullman, and Editors. Database systems: Achievements and opportunities. *Communications of the ACM*, 34(10):110–120, 1991.

[91] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22,224–254, 1964.

[92] D. J. Spiegelhalter. Coherent evidence propagation in expert systems. *The Statistician*, 36:201–210, 1987.

[93] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell. Bayesian analysis in expert systems. Technical Report BAIES Report BR-27, MRC Biostatistics Unit, Institute of Public Health, Cambridge, 1992.

[94] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updates of conditional probabilities on directed graphical structures. *Networks*, 20(579-605), 1990.

[95] R. E. Stepp and R. S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, San Mateo, CA, 1986.

[96] L. Stewart. Hierarchical Bayesian analysis using monte carlo integration: Computing posterior distributions when there are many possible models. *The Statistician*, 36:211–219, 1987.

[97] M. L. Tseng. *Integrating Neural Networks with Influence Diagrams for On-line Sensor Validation and Diagnostic Reasoning.* PhD thesis, University of California at Berkeley, Berkeley, CA, 1991.

[98] W. X. Wen. From relational databases to belief networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 406–413, 1991.

[99] R. L. Wexelblat. On interface requirements for expert systems. *AI Magazine*, 10(3):66–78, 1989.

[100] Q. Wu, P. Suetens, and A. Oosterlinck. Integration of heuristic and Bayesian approaches in a pattern-classification system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* The AAAI Press, Menlo Park, CA, 1991.

[101] Y. H. Wu and S. Wang. Discovering functional relationships from observational data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* The AAAI Press, Menlo Park, CA, 1991.

[102] J. York and D. Madigan. Markov chain monte carlo methods for hierarchical Bayesian expert systems. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 433–439, 1993.

[103] J. M. Zytkow. Combining many searches in the FAHRENHEIT discovery system. In *Proceedings of the Fourth International Workshop on Mahine Learning*, pages 281–287, 1987.

[104] J. M. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases.* The AAAI Press, Menlo Park, CA, 1991.