Ph.D. Proposal, Department of Computer Sciences,
University of Texas at Austin, October 1995.

Refinement of Bayesian Networks by Combining Connectionist and Symbolic Techniques

Sowmya Ramachandran,
Department of Computer Sciences,
University of Texas,
Austin, TX 78712

Supervising Professor: Dr. Raymond J. Mooney

October 20, 1995

Abstract

Bayesian networks provide a mathematically sound formalism for representing and reasoning with uncertain knowledge and are as such widely used. However, acquiring and capturing knowledge in this framework is difficult. There is a growing interest in formulating techniques for learning Bayesian networks inductively. While the problem of learning a Bayesian network, given complete data, has been explored in some depth, the problem of learning networks with unobserved causes is still open. In this proposal, we view this problem from the perspective of theory revision and present a novel approach which adapts techniques developed for revising theories in symbolic and connectionist representations. Thus, we assume that the learner is given an initial approximate network (usually obtained from a expert). Our technique inductively revises the network to fit the data better. Our proposed system has two components: one component revises the parameters of a Bayesian network of known structure, and the other component revises the structure of the network. The component for parameter revision maps the given Bayesian network into a *multi-layer feedforward neural network*, with the parameters mapped to weights in the neural network, and uses standard backpropagation techniques to learn the weights. The structure revision component uses qualitative analysis to suggest revisions to the network when it fails to predict the data accurately. The first component has been implemented and we will present results from experiments on real world classification problems which show our technique to be effective. We will also discuss our proposed structure revision algorithm, our plans for experiments to evaluate the system, as well as some extensions to the system.

Contents

1 Introduction

Theory revision is an area of research that has grown out of work on inductive and explanation-based learning. It is based on the idea that, when only limited data is available, biasing an inductive learner with prior knowledge can improve learning by reducing the search space of possible hypotheses. The more complex the domain, the more the advantage of such a bias. Thus, theory revisions systems assume that the learner has an initial imperfect knowledge base (usually obtained from a domain expert) which is then inductively revised to fit the data. Many techniques have been developed for revising knowledge bases represented in various languages such as propositional Horn-clause logic (Ourston and Mooney, 1994) and relational Horn-clause logic (Cohen, 1992; Pazzani and Kibler, 1992; Richards and Mooney, 1995). Even in the connectionist framework, theory revision has received a fair amount of attention. At the most basic level are the pruning and growing algorithms (Mezard and Nadal, 1989; Mozer and Smolensky, 1989; Frean, 1990; Fahlman and Lebiere, 1989) that use data to either add or delete hidden units from a multi-layer feedforward network. There are also techniques, such as KBANN (Towell and Shavlik, 1994; Opitz and Shavlik, 1993), that explicitly bias a neural network with an initial theory. Experiments on real-world data have demonstrated that revising an approximate domain theory produces more accurate results than learning from training data alone (Ourston and Mooney, 1990; Thompson et al., 1991; Towell et al., 1990).

Research in theory revision has, however, been limited to logical and connectionist representations. Little has been done to address the problem of revising probabilistic knowledge. Intelligent systems need mechanisms to represent and reason with uncertain knowledge. Uncertainty in a domain or a task may arise due to incomplete knowledge about the state of the world or due to true randomness in the domain. Examples of tasks involving uncertainty include medical diagnosis, and plan recognition, to name but a few. Thus, we need languages for representing uncertainty. It is also desirable to have techniques for learning and revising knowledge represented in these languages.

There are many approaches to representing and reasoning with uncertainty, such as *non-monotonic logic*, *fuzzy logic*, *certainty factors*,*Bayesian networks*, and *Dempster-Schafer* calculus. Of these, non-monotonic logic uses a symbolic representation of uncertainty, while the rest use numeric representations. The Bayesian networks approach stands out as the only one that is directly grounded in probability theory, which has long been a widely accepted formalism for representing uncertainty. The rest of the approaches invent their own calculi, which makes their semantics less clearly defined.

Among the numeric representations, *certainty factors* have played a significant role in the history of uncertain reasoning. The use of *certainty factors* is exemplified in MYCIN, an expert system to recommend treatment for bacterial infection (Shortliffe and Buchanan, 1975; Buchanan and Shortliffe, 1984). MYCIN is a rule-based system. However, the rules are augmented with *certainty factors* or numbers that indicate their credibility. The certainty of a conclusion of a rule is computed as a function of the certainty of the premises and the credibility of the rule. Although they have been successfully applied in a number of domains, the rules for combining certainty factors are ad hoc and provide no mathematical guarantees, unless unrealistic independence assumptions are made.

Bayesian networks (Pearl, 1988), on the other hand, represent uncertainties as probabilities of events in the world. They provide ways for representing dependencies between variables explicitly. They also provide theoretically sound mechanisms for combining probabilities, and for accounting for dependencies. Their strong grounding in probability theory makes them a particularly attractive formalism for representing knowledge. Many real-world applications, especially in medical diagnosis, now use Bayesian networks (Pradhan et al., 1994; Burnell and Horovitz, 1995; Fung and Del Favero, 1995). However, like all numerical representation schemes, they suffer from the knowledge acquisition problem. Not only is it difficult to formulate the underlying structure of the network, it is especially difficult to specify the dependencies between the variables in precise numeric terms.

Thus, Bayesian networks provide a mathematically sound language for representing uncertainty, but are hard to acquire. Therefore, it would be useful to have efficient techniques that learn Bayesian networks from data. Not surprisingly, there is now a growing interest in this problem.

Specification of a Bayesian network requires both the specification of the structure of the network, as well as a set of parameters associated with the variables in the network. The details of this will be explained further in the background section. Thus, the task of learning a Bayesian network can be divided into two subtasks: one of learning the structure of the network, and the second of determining the parameters.

Within the general framework of inducing Bayesian networks, we can envision the following scenarios.

1. *Known structure, fully observable*: In this scenario, the structure of the network is completely known and the data includes observations of all the variables in the network. The task here is to learn the parameters of the network from data.

2. *Known structure, hidden variables*: Here, the structure of the network is given and known to be correct. The data, however, does not include observations of every variable in the network. The variables whose observations are not specified in the data are called *hidden variables*. Again, the task is to induce the parameters of the network. However, the task is more complicated in this case due to the presence of hidden variables.

3. *Unknown structure, fully observable*: In this case, neither the structure, nor the parameters of the network are known. We can, however, assume that the only variables in the network are those that are included in the data. The task here is to learn both the structure and the parameters of the network.

4. *Unknown structure, hidden variables*: This is the most general learning scenario where the structure of the network is unknown and there are hidden variables.

The first of these is fairly straightforward. A common approach is to use the maximum likelihood estimates for the parameters, which in the case of no hidden variables, reduces to a function of the relative frequencies of occurrences of the values of the variable (Spiegelhalter and Lauritzen, 1990).

The problem of learning the parameters for a network with a known structure, in the presence of hidden variables, has also received some attention. Many statistical techniques like *Gibbs sampling* (Geman and Geman, 1984) and *EM* (Dempster et al., 1977; Lauritzen,

1995) can be used in the context of Bayesian networks. Russell et al. (1995) have proposed an approach that optimises the probability of the data given the network using a gradient descent algorithm.

The learning problem addressed by Cooper and Herskovits (1992) falls in the third category. Their technique is to use a scoring metric to hill climb through a space of possible Bayesian networks to find one that is the most probable given the data. A number of variations and improvements to this approach have since been proposed (Buntine, 1991; Heckerman et al., 1994; Provan and Singh, 1994).

The fourth scenario above, namely that of learning a Bayesian network with hidden variables and an unknown structure, is by far the most difficult and the least studied. Most of the above techniques could be adapted to discover hidden variables, but at a great cost involving brute force search. Connolly (1993) has proposed using clustering techniques (Fisher, 1987) to discover hidden variables. However, this technique can only learn tree structured networks.

Thus, while researchers have a grasp on some aspects of learning Bayesian networks, the problem of learning Bayesian networks with unknown structures and hidden variables still poses a tough challenge. However, theory revision techniques like those proposed by Ourston and Mooney (1994); Opitz and Shavlik (1993); Mahoney and Mooney (1993a) have been successful in addressing similar issues. The bias that such techniques provide in the form of prior knowledge helps narrow the search space, making these techniques efficient. We had mentioned earlier that there has been very little research into using theory revision techniques to learn Bayesian networks. Lam and Bacchus (1994) have a technique for incrementally refining a Bayesian network using the Minimum Description Length principle (Rissanen, 1978). Buntine (1991) has proposed a technique for revising a Bayesian network efficiently, using scoring metrics similar to that proposed by (Cooper and Herskovits, 1992). However, neither of these techniques can revise networks with hidden variables. Mahoney and Mooney (1993a) have a system for revising probabilistic knowledge bases, but expressed in the form of *certainty factors*. Their system, RAPTURE maps the rules of the knowledge base into a neural network and uses connectionist methods to revise the certainty factors associated with the rules as well as the rules themselves. RAPTURE can also discover hidden variables not specified in the data.

In this research, we are proposing a theory revision approach to learning Bayesian networks that is inspired by RAPTURE. From the perspective of the four learning scenarios outlined earlier, this problem lies somewhere between scenarios 2 and 4. Thus, we assume that the learner is given a Bayesian network that may be incomplete or incorrect. We also assume that the learner is provided with data that may not include all the variables in the network. The task is to use the data to improve the predictive accuracy of the network.

Our system for revising Bayesian networks, called BANNER (BAyesian Networks NEural Revision), is based on adapting symbolic and connectionist theory revision techniques similar to EITHER and KBANN (Ourston and Mooney, 1994; Opitz and Shavlik, 1993). These techniques provide a way to determine when the theory needs to be revised. They also cut down the search space by focusing attention on local portions of the network that should be modified. Our research, we hope, will lead to an efficient technique for revising a Bayesian network. Another significant contribution of our research would be to provide a way to invent hidden causes efficiently.

Whenever the structure of a network is changed, the parameters have to be modified as well. In this proposal, we also describe a system that learns the conditional probabilities for a network with *noisy-and* and *noisy-or* nodes by mapping such a network onto a *multi-layered feed-forward neural network (ANN)* and refining the weights using standard backpropagation techniques. This enables the learning of conditional probabilities even in the presence of hidden variables.

We will first present a brief introduction to the various concepts that are central to our research. Next, we will present our approach to the problem of revising Bayesian networks. We will also present the results from our preliminary experiments with BANNER. We will conclude with a discussion of our proposed future research.

2 Background

2.1 Bayesian Networks: An Overview

Bayesian networks (Pearl, 1988) provide a formalism for representing probabilistic knowledge. In general, a Bayesian network is a directed acyclic graph whose nodes correspond to random variables. These variables can take on many values. In this paper, an upper-case letter represents a variable and the corresponding lower-case letter represents the value associated with the variable. A variable X is called a *parent* of another variable Y, if there is a directed link from X to Y. In such a case, Y is called the *child* of X. A variable is called a *spouse* of another variable, if the two share a common child. For example, in the Bayesian network shown in Figure 1, B is the parent of A, A is the child of B, and E is the spouse of B. Associated with each node is a *conditional probability table (CPT)*, which gives the probability of each value of the variable given each possible combination of values of its parent nodes. Variables can be discrete or continuous. Given a network with n nodes and the associated CPTs, the probability of a conjunction of a particular assignment of values to the variables, i.e. $P(x_1, \ldots, x_n)$, can be calculated using the following formula:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(X_i)) \tag{1}$$

where $P(x_i \mid Parents(X_i))$ is obtained from the CPT associated with variable X.

Figure 1 (Pearl, 1988) shows an example of a Bayesian network. Such a network maybe used by a person to decide whether or not to respond to a alarm in his house. The nodes in the network represents the various events that are of relevance to the decision. Thus, the node E represents the occurrence of an earthquake, node R represents the announcement of an earthquake on the radio, and node D represents the event of the person's daughter calling him about the alarm. The links between the nodes represents the dependencies between the various events. For instance, the network indicates that A is conditionally dependent on B and E, while G is independent of B given A. The links can also be seen to indicate *causality*. Thus, the link from *earthquake* to *alarm* can be interpreted as a statement that earthquake *causes* the alarm to go off. The CPT associated with each variable reflects the strength of the causal influence on the variable from its parents. For example, the CPT associated with variable A in Figure 1 would specify:

$$M(A) = \left[\begin{array}{cccc} P(A \mid \neg B, \neg E) & P(A \mid B, \neg E) & P(A \mid \neg B, E) & P(A \mid B, E) \\ P(\neg A \mid \neg B, \neg E) & P(\neg A \mid B, \neg E) & P(\neg A \mid \neg B, E) & P(\neg A \mid B, E) \end{array} \right] \quad (2)$$

Given the CPTs, the joint probability distribution of the variables in the network can be computed as follows:

$$P(b, e, r, a, d, w, g) = P(b)P(e)P(r \mid e)P(a \mid e, b)P(d \mid a)P(w \mid a)P(g \mid a) \quad (3)$$

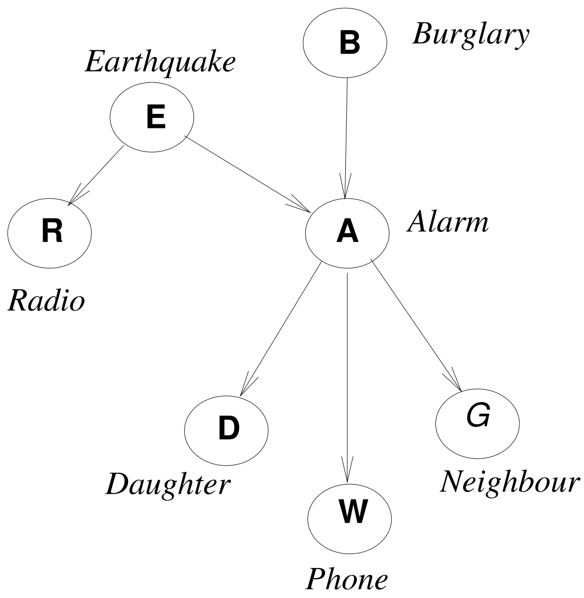

Figure 1: Bayesian Network - Example

Typically, a Bayesian network is used to infer the probability distribution of a set of variables T, given the value of another set of variables E (called *evidence*). This inference can be *predictive* (inferring effect from causes). For example, one might want to use the network in Figure 1 to infer the probability that his daughter will call, given evidence that a burglary has occurred. Sometimes, the inference is *diagnostic* or *abductive* (inferring causes from effects). This is the case, for instance, when one uses the network to infer the probability of a burglary, given the evidence that his daughter called him to report the alarm. It is also possible to combine predictive and diagnostic inferences. Thus, one might want to infer the probability of the alarm going off, given the evidence that no burglary was known to occur, but that the radio announced an earthquake. The complexity of the inference procedure depends on the structure of the network. For a network that is a *polytree*, i.e. in which each pair of variables is connected by at most one undirected path, the inference procedure is linear in the size of the network. For networks with undirected loops, however, the inference is NP-hard (Cooper, 1990).

The specification of a general Bayesian network is combinatorial in the fan-in of the nodes. It requires the specification, for each variable, of the conditional probabilities of the variable given all possible combinations of values of its parents. Thus, for a network where all variables are binary-valued, a variable with n parents would require 2^n conditional probabilities to be specified.

The *noisy-or* and the *noisy-and* models of Bayesian networks (Pearl, 1988) avoid this problem by providing a way to compute the conditional probability of a variable given a combination of values of its parents from just the conditional probabilities of the variable given the value of each of its parents in isolation. In the following subsections, we further elaborate on the specific combination rules provided by each of these models.

2.1.1 The noisy-or model

A *noisy-or* node in a Bayesian network is a generalisation of a logical *or*. As in the case of the logical *or*, an event E is presumed to be false if all the conditions that cause E are false (i.e. P(E)=0). However, unlike a logical *or*, if one of the causes of the event E is true, it does not necessarily imply that E is definitely true. Each condition C_i causing the event E can be thought of as having an associated inhibitory influence which is active with a probability q_i. Thus, if C_i is the only cause of E that is true, then E is true with a probability $(1 - q_i)$. Moreover, the likelihood of E being true is a monotonic function of the number of its causal conditions that are true. The parameter $c_i = 1 - q_i$ is the degree to which an isolated cause C_i of an event E can endorse the event.

Given some evidence, the c_i associated with each link in a network, and the belief measures of all the parents of a node in the network, there is a simple equation for calculating the *degree of belief* that the node is true. Under the assumption that all the evidence in the network is causally upstream of the node, the degree of belief in a node X is given by

$$Bel(x) = \begin{cases} \prod_i (1 - c_i \pi_{iX}) & \text{if } x = 0 \\ 1 - \prod_i (1 - c_i \pi_{iX}) & \text{if } x = 1 \end{cases} \tag{4}$$

where π_{iX} is the degree of belief in the truth of the i-th parent of X. Thus, the number of parameters that have to be specified for a *noisy-or* node is linear in its fan-in.

2.1.2 The noisy-and model

A *noisy-and* node is the dual of a *noisy-or* node. It is a generalisation of a logical *and*. As in the case of a logical *and*, an event E is presumed to be true if all the conditions that cause E are true (i.e. P(E)=1). However, unlike the logical *and*, if one of the causes of the event E is false, it does not imply that E is definitely false. Each condition C_i causing the event E can be thought of as having an associated enabling influence which is active with a probability q_i. Thus, if C_i is the only cause of E that is false, then E is false with a probability $(1 - q_i)$. Moreover, the likelihood of E being false is a monotonic function of the number of its causes that are false. The parameter c_i is the degree to which disproving an isolated cause of an event disproves the event itself.

The belief measure of a *noisy-and* node X, given some evidence, the c_i associated with each link in the network, and the belief measures of all the parents of the node, is given by

$$Bel(x) = \begin{cases} 1 - \prod_i (1 - c_i(1 - \pi_{iX})) & \text{if } x = 0 \\ \prod_i (1 - c_i(1 - \pi_{iX})) & \text{if x = 1} \end{cases} \tag{5}$$

where π_{iX} is the degree of belief in the truth of the i-th parent of X. Here again, the assumption is that all the evidence in the network is causally upstream of the node.

2.1.3 Polytree Vs. Loops

The structure of a Bayesian network has a significant influence on the complexity of inference. Based on their structure, Bayesian networks can be divided into two classes: *polytrees*, which have a simple structure where each pair of variable is connected by at most one path, and *networks with loops*, which have undirected loops in the structure. The inference algorithm for polytrees is linear in the size of the network and the propagation rules for beliefs are local. However, the presence of loops increases the complexity of inference significantly. The derivation of the propagation rules for polytrees exploit the fact that the network is singly connected. These rules cannot be used in the presence of loops.

Several algorithms have been proposed for handling loops (Pearl, 1988). *Clustering* is a technique where the variables forming a loop are clustered into one node, which results in a polytree. Inference is done using the polytree algorithm on the clustered network. The clustered nodes are then separated into individual nodes whose beliefs are computed from the belief of the cluster. *Conditioning* is a technique which uses case analysis to propagate beliefs. The algorithm picks a variable from each loop and this set of variables is instantiated to all the possible values the variables can take. Instantiating a variable in each loop breaks the loops and the network reduces to a polytree. Beliefs are propagated for each of these cases using the polytree algorithm. The overall belief of each node is computed as the weighted average of the belief computed for the node for each of the cases. *Stochastic simulation* is an algorithm for simulating the network starting at some random state. The belief associated with each variable is the frequency of occurrence of each value of the node over a large number of simulations.

2.2 Multi-layered Feed-Forward Neural Networks

Our approach uses neural networks, specifically, multi-layered feed-forward networks, to revise Bayesian networks. Here, we describe these networks briefly.

Figure 2 shows an example of a multi-layered feed-forward neural network (Hertz et al., 1991). The output units are denoted by O_i and the input units by I_k. The layer between the input and the output layers is the hidden layer, whose units are denoted by H_j. The connections between the units in the input and the hidden layers are denoted by w_{ij} and those between the hidden and output layers are denoted by W_{jk}.

When the input units of a network are clamped to particular values, the output of the network is computed by propagating these values through the hidden layers on to the output layer. The activation of each unit is a function of a weighted combination of the values of all the units feeding into it. Typically, each unit computes a function of the weighted sum of all its inputs, as shown below.

Given a pattern μ, the net input received by each hidden unit j is given by

$$h_j^\mu = \sum w_{jk} I_k \tag{6}$$

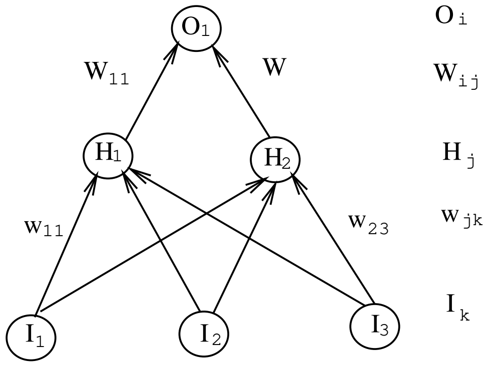

Figure 2: Multi-layered Feed-Forward Neural Network - Example

The activation or the output of each hidden unit is given by

$$V_j^\mu = g(h_j^\mu) \tag{7}$$

where $g(h)$ is a thresholding function, or a continuous sigmoid function. The net inputs and the activations of the output units are computed similarly.

Given some data specifying a set of desired values for the input and output variables, the network can be trained on the data using the backpropagation algorithm. This algorithm uses a gradient descent approach to modify the weights in the network so as to minimize the error of the network on the data. The error measure is defined by

$$E[\mathbf{w}] = 1/2 \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 \tag{8}$$

where ζ_i^μ is the desired value of the i-th output variable for pattern μ of the data and O_i^μ is the actual output of the i-th output unit for the same pattern.

The backpropagation algorithm then changes each weight w_{ij} by an amount Δw_{ij} proportional to the gradient of E given by:

$$\Delta w_{ij} = -\eta \partial E / \partial w_{ij} \tag{9}$$

where η is a parameter that controls the *learning rate*. Deriving an accurate gradient of the error function is, therefore, a crucial aspect of the backpropagation approach.

At the start of training, the network is initialised with some (typically random) weights.[1] For each pattern in the data, the inputs of the network are clamped with the input values specified in the pattern. These values are propagated forward to the output units. The error between the output values predicted by the network and those specified in the data is propagated back through the hidden layers to the input layer. The error propagated to each unit is used to modify the weights of the connections feeding into the unit, according to the learning rule given by Equation 9. This process is repeatedly iterated over all the patterns in the data until a desired level of convergence is reached. It should be noted that it is possible for the network to settle into a local minimum. Depending on the initial weights, it is also likely that the training will not converge. It is often necessary to fine tune the training parameters, such as the learning rate, to get convergence.

[1] Sometimes the initial weights are determined using domain knowledge.

3 Our Approach

Our approach to the problem of revising Bayesian networks has grown out of the following insights.

- There are similarities between Bayesian networks and neural networks in that both use local computations for inference. The value associated with each node (*activation* in the case of an ANN, *degree of belief*s in a Bayesian network) can be computed just from the values of neighboring nodes and the strength of the connections with them. Thus, variables in a Bayesian network are analogous to nodes in a neural network. The conditional probabilities in a Bayesian network are analogous to the weights in a neural network. This suggests that methods similar to gradient descent can be used to learn the parameters of a Bayesian network.

- Most of the existing approaches to learning Bayesian networks use heuristics to optimise the probability of the data given a network. However, most often a Bayesian network is built for a specific task of prediction or diagnosis. We believe that a network trained for optimal performance in its intended task will fare better than one trained to optimise the probability of the data given the network. Thus, a network that will be used for prediction should be trained to maximise its prediction accuracy. Intuitively, training a network for a specific task would require less data than training it for all possible tasks. Of course, the validity of this claim will have to be demonstrated empirically.

- Most of the techniques discussed in the previous sections focus on the probabilistic aspects of a Bayesian network. Since probability distribution is a global property, these techniques cannot detect the incorrectness of a Bayesian network locally. Nor can they focus on local portions of the network that require revision. Symbolic theory revision techniques, on the other hand, localise the blame for incorrect predictions on portions of the theory which are then revised. This restricts the search space considerably. If, instead of viewing a Bayesian network as a quantitative model, we could look at it as a specification of qualitative relationship among variables (Wellman, 1990), we would be able to apply symbolic theory revision techniques to revise the Bayesian network locally. The idea is to reduce the search space of possible revisions by focusing attention on a subset of variables that can be held responsible for an erroneous prediction by the network.

Here, we propose a two-tiered approach, similar to RAPTURE (Mahoney and Mooney, 1993a; Mahoney and Mooney, 1994), to the problem of learning a Bayesian network from partial specification. Given some data, we first try to improve the Bayesian model by revising the parameters of the network. If the network still does not fit the training data, then the structure of the network is modified to find the network with the highest predictive accuracy. Thus, our approach has two distinct components:

1. The first component assumes that the underlying Bayesian structure is correct and is concerned with modifying the parameters of the network to improve predictive accuracy. For this task, we first map the Bayesian network into a neural network and use the standard backpropagation algorithm to optimise the parameters.

2. The second component is concerned with modifying the structure of the network. To do this, we approximate the Bayesian network by a *Qualitative Probability Network (QPN)* (Wellman, 1990). Given a set of data which are predicted incorrectly by the network, we use qualitative analysis to determine the portion of the network that needs to be revised.

Figure 3 illustrates the interactions between these components. Note that their modularity and separation allows us to replace the algorithm implementing each component without affecting the entire system.

This division of the overall problem suggests that our research should proceed in two stages: the first stage which investigates techniques for learning parameters, and the second stage that investigates the problem of structure revision. In fact, this has been our approach. In the following section, we will first discuss our research into parameter revision techniques, followed by a discussion of our proposed research into structure revision.

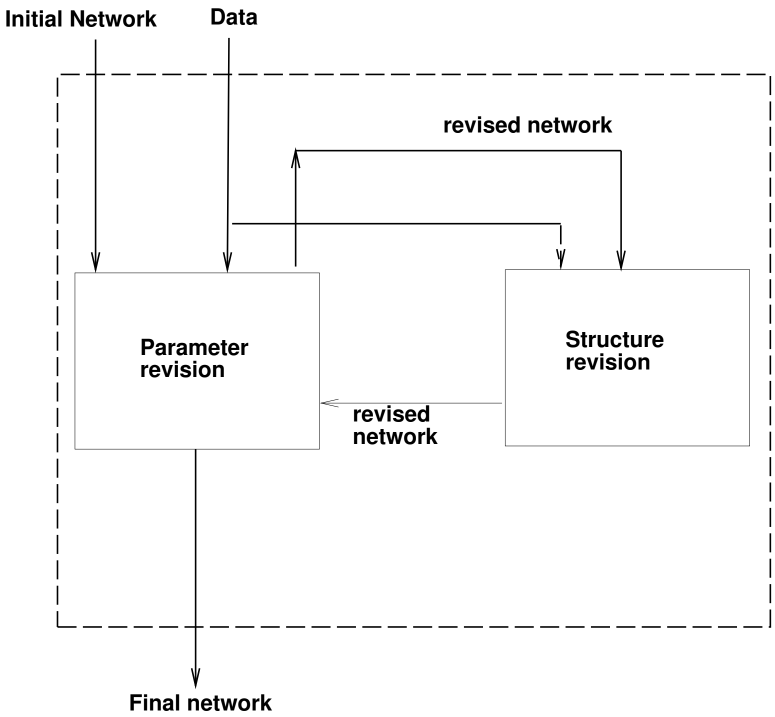

Figure 3: Overview of Our System

4 Parameter Revision: Learning Conditional Probabilities

In this section, we present our on-going research into techniques for parameter revision. We approach the problem by first investigating the simpler case i.e. where it is assumed that network is being trained for predictive inference. We then address the general case where the network is trained for a combination of predictive and diagnostic inferences. For each of these cases, we present our approach followed by some experiments that evaluate its performance, as well as a discussion of future research.

Our proposed technique for learning the conditional probabilities on a Bayesian network works as follows:

1. Map the given Bayesian structure onto a multi-layered feed-forward neural network.

2. Train the neural network on the given data using the standard gradient descent back-propagation algorithm.

3. Map the trained neural network back onto a Bayesian network.

While this top level description of the algorithm is general and covers all inference scenarios, we have found the details of the algorithm for the predictive case to be different from those for the general case. Due to differences in the computations involved in the two cases, the learning rules for the predictive case are much simpler.

4.1 Parameter Revision for the Predictive Case

4.1.1 Neural Network Mapping

Recall that the first step in our technique is to map the Bayesian network into a neural network. The structure of the ANN is identical to the structure of the given Bayesian network. A *noisy-or* node in the Bayesian network is mapped onto a *noisy-or* unit in the neural network. A *noisy-and* node in the Bayesian network is mapped onto a *noisy-and* unit in the neural network. The causal link between the nodes of the network are mapped onto connections between the corresponding units in the ANN. The weight on each link corresponds to the c_i associated with the link in the corresponding Bayesian network (Section 2.1).

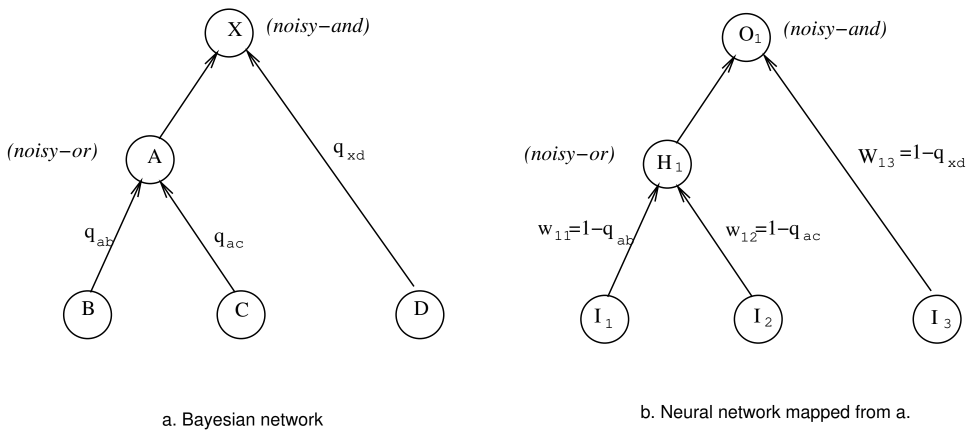

Figure 4: Mapping from a Bayesian network into a neural network

Thus, the Bayesian network shown in Figure 4a is mapped on to an ANN shown in Figure 4b. The output of the *noisy-or* and *noisy-and* units are computed using the following functions:

$$activation(i) = \begin{cases} 1 - \prod_j (1 - w_{ij} O_j) & (\textit{Noisy-or}) \\ \prod_j (1 - w_{ij}(1 - O_j)) & (\textit{Noisy-and}) \end{cases} \tag{10}$$

where
O_j is the activation of the jth unit feeding into unit i,
w_{ij} is the weight of the link between unit i and the jth unit feeding into it.

This function computes the degree of belief in the truth of the variable represented by the unit. Note that, since the only evidence placed in the network is on the input variables, the assumption that all evidence in the network is causally upstream of all the hidden and output variables holds.

Since the mapping is direct, it is straight forward to recover the Bayesian network from a trained ANN.

4.1.2 The Training Phase

Once the Bayesian network is transformed into an ANN as described in the previous section, the problem of learning the parameters (the c_i's) of the Bayesian network is transformed into a problem of learning the weights in the ANN. This is achieved by training the ANN using standard backpropagation techniques.

The data used for training the network should consist of a set of patterns that specify the value for the input and the output variables of the network. Since we are assuming that all the evidence in the network is causally upstream of all the hidden and output variables, input variables are the *sources* in the DAG representing the network (i.e those variables that have no incoming links). The *sinks* in the DAG are the output variables. The values for the hidden variables do not have to be specified in the data.

To train the network, the backpropagation algorithm requires a learning rule. These are derived by incorporating the activation function for a *noisy-or* or a *noisy-and* unit (Equation 10) into the error function defined in Equation 8 and taking the partial derivative of the resulting error function with respect to the weights, as per Equation 9. This results in the following learning rules for the *noisy-or* and *noisy-and* units:

$$\Delta w_{ij} = \begin{cases} \eta \delta_i O_j \prod_{k \neq j} (1 - w_{ik} O_k) & (\textit{Noisy-or}) \\ -\eta \delta_i (1 - O_j) \prod_{k \neq j} (1 - w_{ik}(1 - O_k)) & (\textit{Noisy-and}) \end{cases} \tag{11}$$

where
η = learning rate and
δ_i = error propagated back from the output units to unit i. O_j is the activation of the jth unit feeding into i,
w_{ij} is the weight of the link between unit i and the jth unit feeding into it.

Once the network has been trained to a desired accuracy following the procedure described in Section 2.2, it can be mapped back into a Bayesian network.

4.1.3 Experimental Evaluation

We have evaluated Banner on two classification problems: DNA promoter recognition (Noordewier et al., 1991) and DNA Splice Junction recognition (Noordewier et al., 1991). Each of these has associated with it an initial domain theory that does not have a good prediction accuracy on the data and therefore has to be revised.

For each problem, we created several random splits of the data into training and test sets. One of these data splits was used to determine the stopping point for training. The network was trained until further training did not decrease the *root mean square error* of the network on the training set. The networks for the remaining splits were trained for the same number of epochs as the first one.

Here, we present the results of our experiments. We also compare the performance of BANNER with the performances of other learning systems like KBANN, ID3, EITHER, RAPTURE and BACKPROP. ID3 (Quinlan, 1986) is a system for inducing decision trees. EITHER (Ourston and Mooney, 1994) learns and revises propositional Horn-clause theories. RAPTURE (Mahoney and Mooney, 1993b) is a system for revising certainty-factor rule bases using neural networks. KBANN (Towell et al., 1990; Noordewier et al., 1991) revises a logical theory using a hybrid of symbolic and connectionist learning methods. BACKPROP is the standard backpropagation approach using a three-layer feedforward neural network.

DNA Promoter Recognition: Figure 5 shows the initial logical theory for recognising a DNA promoter sequence. There are 57 input features called nucleotides, each of which can take on one of four values, A, G, T and C. The target class, *promoter*, predicts whether or not the input DNA sequence indicates the start of a new gene.

```
promotor <– contact, conformation

contact <– minus_35, minus_10

minus_35 <– (P–36 T), (P–35 T), (P–34 G), (P–33 A), (P–32 C).

minus_35 <– (P–36 T), (P–35 T), (P–34 G), (P–32 C), (P–31 A).

minus_10 <– (P–14 T), (P–13 A), (P–12 T), (P–11 A), (P–10 A), (P–9 T).

minus_10 <– (P–13 T), (P–12 A), (P–10 A), (P–8 T).

minus_10 <– (P–12 T), (P–11 A), (P–7 T).

conformation <– (P–47 C), (P–46 A), P(–45 A), (P–43 T), (P–42 T), (P–40 A)
                (P–39 C), (P–22 G).

conformation <– (P–45 A), (P–44 A), (P–41 A).

conformation <– (P–49 A), (P–44 T), (P–27 T), (P–22 A), (P–18 T), (P–16 T),
                (P–15 G), (P–1 A).

conformation <– (P–45 A), (P–41 A), (P–28 T), (P–27 T), (P–23 T), (P–21 A),
                (P–17 T), (P–4 T).
```

Figure 5: DNA Promoter Recognition - Initial Domain Theory

Figure 6 shows a portion of the Bayesian network corresponding to this theory. All the logical *and*s in the domain theory were mapped onto *noisy-and* nodes and all the logical *or*s are mapped onto *noisy-or* nodes. Each 4-valued input feature has been converted into *four* binary-valued features.[2] This network was translated into a neural network as described

[2]We use binary-valued nodes because the noisy-or and noisy-and nodes are binary-valued. However,

earlier. The initial weights were all set to random values close to 1.0 to mimic the initial logical theory. It is necessary to perturb the weights around the value 1.0 to break the symmetry for better convergence.

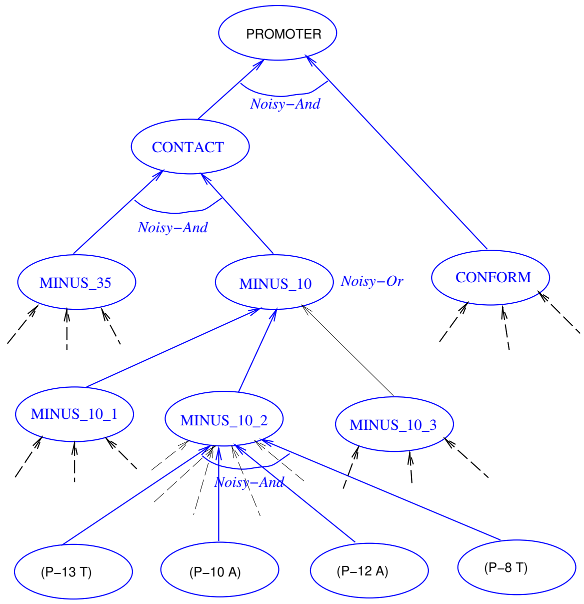

Figure 6: DNA Promoter Recognition - Bayesian Network

The data consisted of 106 patterns (53 positive and 53 negative examples). Figure 7 shows the learning curve determined from this experiment. This graph is a plot of the average accuracy of the network at classifying DNA strings over 25 different random splits. It clearly demonstrates that our technique is successful in improving the accuracy of the network substantially (by about 40 percentage points).

The graph also shows the performance of some of the inductive learning algorithms (ID3 and BACKPROP) and theory revision algorithms (EITHER, RAPTURE and KBANN) on the same task. The theory revision systems started out with the same initial theory as BANNER(Figure 5), which they subsequently revised to fit the data. Our technique performs better than ID3, EITHER, and BACKPROP. Its ultimate performance is comparable to both RAPTURE and KBANN, although its learning curve is not as steep.

DNA Splice Junction: We have also evaluated BANNER on the task of learning to recognise the *splice junctions* in a given DNA sequence (Noordewier et al., 1991). Human DNA sequences consist of two regions: *extron* regions encoding information that is used for protein synthesis, interspersed with *intron* regions which are "garbage". The junctions between these regions are called *splice junctions*. There are two kinds of splice junctions: IE sites which are at exon←intron boundaries, and EI sites which are at exon←intron boundaries. These form the two output categories for the classification problem. There are 60 input features, each of which represents a nucleotide and can take on the values A, C, G or T. The domain theory

there have been recent extensions that allow multi-valued noisy-or nodes. We will extend our technique to accommodate multi-valued features in the future.

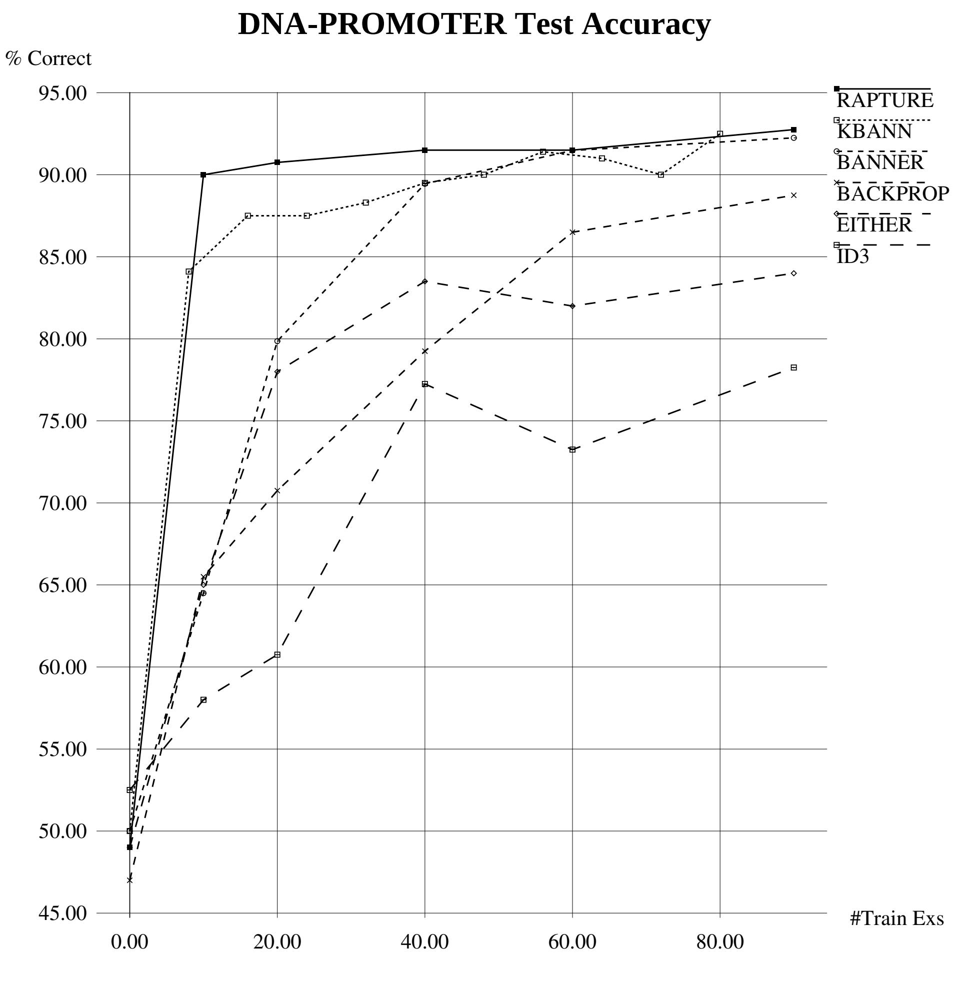

Figure 7: DNA Promoter Recognition - Prediction Accuracy

specifies rules for recognising these sites from patterns of nucleotides around them. Here, we won't go into the details of the domain theory except to mention that it uses M-of-N style rules, where if "M" of the "N" antecedents of a rule are true, then the consequent in considered true. The initial logical domain theory was converted into a Bayesian network and subsequently into a neural network as described in the previous subsection. The weights in the neural network were initialised to random values close to 1.0 to mimic the logical theory.

The data consisted of 2190 patterns, of which we randomly selected 900 patterns which were then divided into training and test sets. Figure 8 shows the learning curves for BANNER and some of the other inductive learning algorithms. The graph shows that the performance of BANNER is comparable to that of KBANN and BACKPROP. Although RAPTURE and KBANN perform better than BANNER it should be noted that they modify the structure of the domain theory as well, which BANNER is not yet equipped to do.

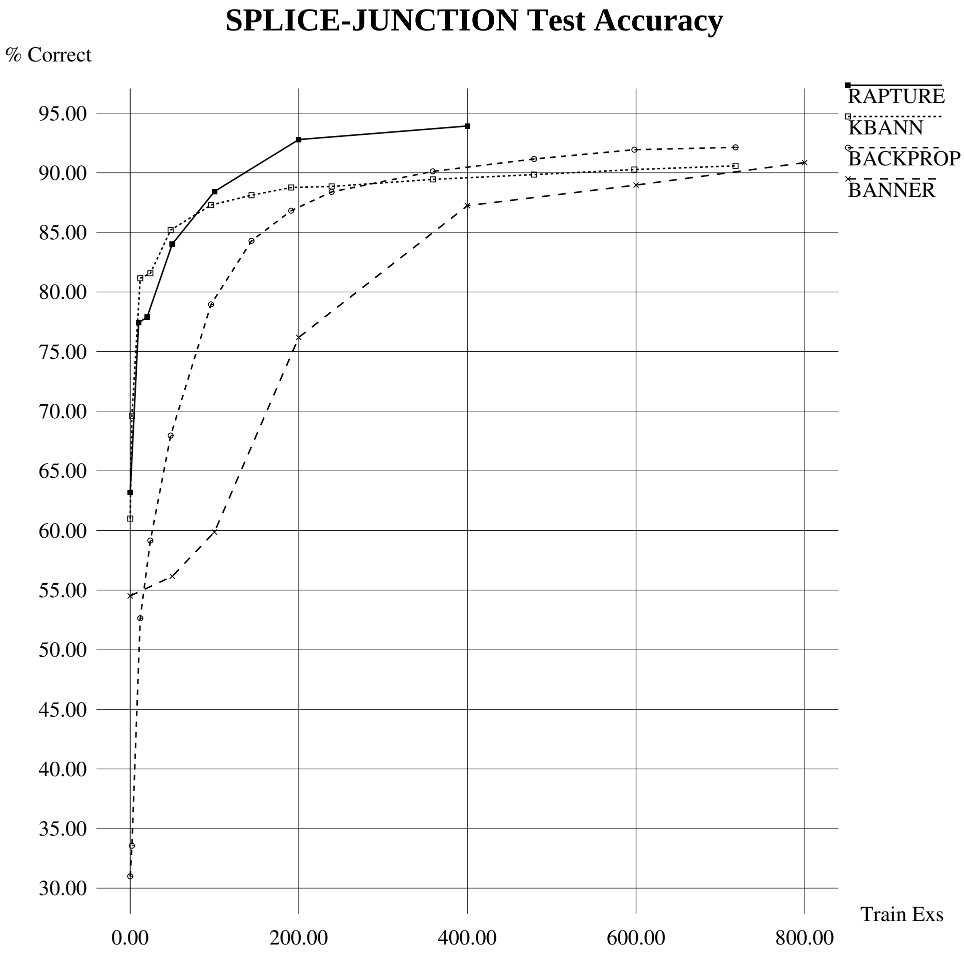

Figure 8: Splice Junction - Prediction Accuracy

4.1.4 Future Work

So far we have shown Banner to be effective on the two standard learning problem. Although its learning rate is slower, Banner eventually performs comparably with other inductive learning systems. We would like to run similar experiments to compare the performance of our system with other systems that learn Bayesian networks with hidden variables, such as the *Adaptive Probabilistic Networks* (Russell et al., 1995) approach and the *EM* (Dempster et al., 1977; Lauritzen, 1995) algorithm. Our experiments with learning parameters for the general case (McQuesten, 1995) suggest that non-gradient techniques using *simulated annealing* (Hertz et al., 1991) are more effective than backpropagation. We also propose to compare the non-gradient training regimes with backpropagation for the predictive case.

One limitation with the current approach is that it cannot be used with networks that have undirected loops. As discussed in Section 2.1.3, loops increase the complexity of inference significantly. Moreover, the computations involved in the inference procedure are more complex and less local than those for polytrees. Since our approach relies on the locality of the computations and involves computing the gradient of the belief functions, it is significantly harder to extend this approach to handle networks with loops.

Non-gradient techniques, however, which only require that the beliefs for each variable be computable, but are not sensitive to the actual computation, can be used even with networks with loops. This is an additional advantage of using such techniques and we propose to investigate this approach as a part of our research.

4.2 Parameter Revision for the General Case

In the previous subsections, we discussed a technique for learning the conditional probabilities on a network using the gradient descent backpropagation algorithm. In formulating the activations and the learning rules, we assumed that all the evidence was at the causal end of the network. This learning rule thus optimises the network for predictive tasks. However, many belief networks are built for abductive reasoning and some use a combination of predictive and abductive reasoning. In order learn the parameters for these networks, we have to relax the assumption about the placement of evidence. Although the basic idea of using backpropagation to learn the conditional probabilities remains the same, the activations and the learning rules are more complicated for the general case where no assumptions about the placement of evidence are made.

Our approach, for the general case, follows the same steps as for the predictive case: the network is first mapped onto a neural network, the network is trained using the backpropagation algorithm, and the trained network is mapped back into a Bayesian network. The procedure for mapping the Bayesian network onto a neural network is exactly the same as that for the predictive case. The difference lies in the details of training, which proceeds as follows:

1. For each training case, place the evidence on the network and propagate the beliefs using standard Bayesian inference algorithms.

2. Once the final belief associated with each variable is established, use the learning rules to propagate the errors back from the target variables to the rest of the network.

3. Use the error accumulated on each variable to modify the conditional probabilities associated with the variable.

Bayesian network algorithms distinguish between *diagnostic evidence* and *predictive evidence*. Figure 9 shows a node X with a set of parents U, and a set of children Y. The evidence in the network e can be partitioned into a set e_X^- of all the evidence at the head of the link $X \rightarrow Y$, and the set e_X^+ of all the evidence at the tail of the link $X \rightarrow U$. The degree of belief associated with the values of variable X is given by:

$$BEL(x) = \alpha\lambda(x)\pi(x) \tag{12}$$

where

$$\lambda(x) = P(e_X^- \mid x)$$

$$\pi(x) = P(x|e_X^+)$$

and α is a normalising constant.

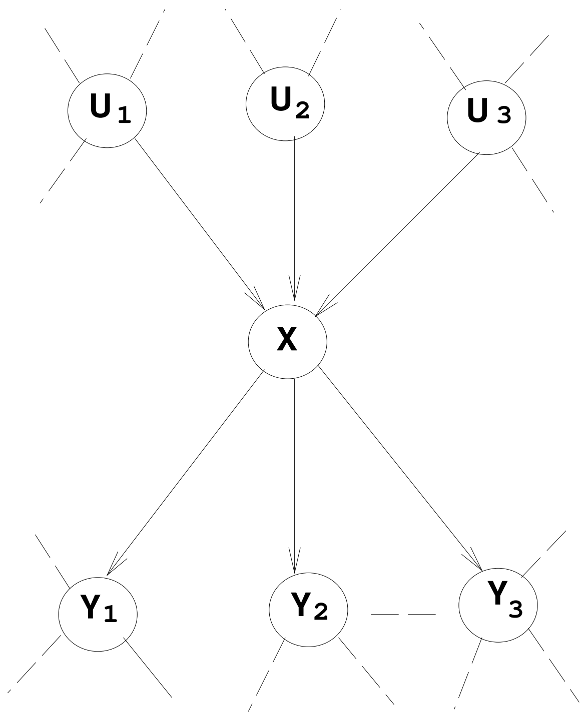

Figure 9: A Fragment of a Polytree

The diagnostic evidence, λ, for node X comes from the variables at the head of the link $X \rightarrow Y$. The predictive evidence for X, π, comes from the tail of the link $U \rightarrow X$. Note that, in a polytree, a node may receive diagnostic evidence from its spouse and predictive evidence from its siblings.

In the predictive case, when all the evidence is on the root nodes of the network, the λ for all nodes is 1, since there is no diagnostic evidence. This reduces the belief computations to a propagation of the π values. However, in the general case, the λs do come into play while calculating the beliefs. Even in the specific case when all the evidence in the network is diagnostic (i.e. on the *sinks* of the network), neither the πs nor the λs reduce to unity and therefore must be considered in the computation.

The first step consists of multiple passes of λ and π propagations before the final belief of each node can be computed. In the second step of the algorithm, the errors that get propagated have two components: the λ *errors*, and the π *errors*. These errors gets propagated throughout the network following an algorithm similar to that used in propagating the beliefs. Once the error propagation attains quiescence, the changes to the parameters in the link are computed based on the error accumulated in each node.

The λ errors and the π errors are computed as derivatives of the belief propagation functions. These gradients and the learning rules for modifying the weights based on the errors are presented in (McQuesten, 1995).

4.2.1 Experimental Evaluation

McQuesten (1995) also reports on experiments conducted to evaluate this approach. We picked a small network for our preliminary experiments designed to give us a quick evaluation of our approach. We used the network shown in Figure 1 as our test bed. We used IDEAL (Srinivas and Breese, 1993), a system for reasoning with Bayesian networks, to simulate the network and generate data. This data was then given to our system, along with the structure of the network, to learn the parameters.

In all of our experiments, success was uniformly defined as mean square error less than 10^{-6}. The backpropagation experiments were run in the batch mode, i.e. the weight changes were accumulated and applied at the end of each training epoch.

We ran several trials in which we trained networks that were initialised with random weights. Our backpropagation algorithm succeeded only twice out of 60 trials with a limit if 2000 epochs. There were no successes in 28 runs to 5000 epochs, nor in 3 runs to 10,000 epochs.

We also ran some experiments where we used *simulated annealing* (Hertz et al., 1991), instead of backpropagation, to train the network. This training regime does not require the gradient of the error function in order to modify the weights. Instead, it makes random perturbations to the weights of the network and picks the best network in each iteration. In experiments conducted with the same data set as above, the technique converged successfully in 16 out of 25 trials on networks with randomly initialised weights.

Thus, our preliminary experiments indicate that, although the backpropagation technique does improve the training performance of the network (indicated by falling mean squared errors), it does not converge quickly. This suggests that we should run more rigorous experiments with different learning rates to study the rate of convergence. We will also analyse the causes that lead to poor convergence. Such an analysis may suggest some improvements that can be made to the training algorithm. Finally, we will also investigate the faster training algorithms such as the *conjugate gradient* (Hertz et al., 1991) approach.

4.2.2 Future Work

Our experiments, indicating the success of simulated annealing techniques in learning the parameter, is especially interesting since this training regime does not require gradients and hence does not depend on the computations involved in the propagation of activations over the network. This will enable us to build a parameter revision subsystem that works uniformly with the different kinds of reasoning mechanisms, abductive and deductive. This will also enable our system to handle networks with loops. Our results are, however, preliminary and we will have to perform more rigorous experiments to show that simulated annealing does indeed yield good results.

5 Structure Revision

When the revision of conditional probabilities on a network fails to improve its predictive accuracy, the network is passed on to the structure revision algorithm. In order to revise the structure of a network, we take a qualitative view of the network. Starting with the target variables that are misclassified, we use qualitative analyses to hypothesise revisions to the network. We use the ideas of *Qualitative Probability Networks* (QPN), presented in Wellman (1990) in our analyses. We first describe QPNs and then give an outline of our structure revision algorithm.

5.1 Qualitative Probability Networks

Qualitative Probability Networks (Wellman, 1990) are an abstraction of graphical representations of probability distributions such as Bayesian networks. They represent the dependencies between variables in terms of qualitative relationships rather than numerically. Qualitative relationships specify constraints on the joint probability distribution over the variables and are of two types: *qualitative influences* and *qualitative synergies*. Although Wellman (1990) describes how these qualitative relationships could be used for inference, in our research we are more interested in the qualitative relationships themselves rather than the inference procedures. In this section, we restrict ourselves to a discussion of the qualitative relationships.

Figures 10 and 11 show the qualitative abstractions of the *noisy-or* and the *noisy-and* gates respectively. The signs on the arrows between the variables indicate qualitative influences and the signs in the boxes indicate qualitative synergies.

Qualitative influences specify the direction of the relationship between two variables. The possible influences between variables are $+$, $-$, 0, ?. A positive influence ($+$) between a variable A and its child B implies that higher values of B are more likely, given higher values of A (this assumes that the values associated with a variable are ordered). Thus, in the binary case, variable B is more likely to be true, given higher likelihoods of A being true. A negative influence is similarly defined. A 0 influence signifies a lack of qualitative influence between two variables and the ? signifies an unknown influence.

Qualitative synergies describe the interactions among influences. For example, if two variables A and B influence a variable X, then qualitative synergies would describe how the influence of A on X affects the influence of B on X. Intuitively, two influences interact *subsynergistically* when increasing the likelihood of one parent has less effect when the other

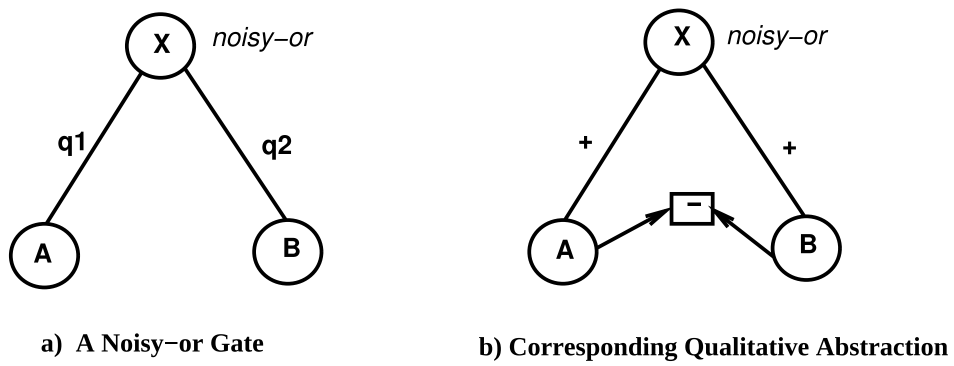

Figure 10: Qualitative Abstraction of a Noisy-Or Gate: The box with the - indicates a subsynergistic relationship

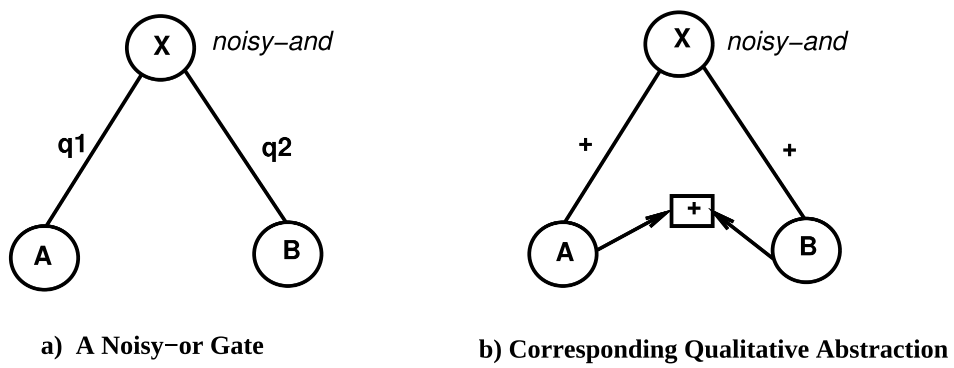

Figure 11: Qualitative Abstraction of a Noisy-And Gate: The box with the + indicates a synergistic relationship

already has a high likelihood. The *noisy-or* gate is an example where influences interact sub-synergistically. Thus, in figure 10b, the higher the likelihood of B, the lesser the influence of A on X. The influences involved in a *noisy-and* gate, on the other hand, interact *synergistically*. In this case, raising the likelihood of one parent has more effect when the other parents have a high likelihood themselves. Thus, in figure 11b, increasing the likelihood of A has the effect of increasing the influence of B on X.

5.2 Outline of the Structure Revision Algorithm

A Bayesian network is typically built for inferring the values of certain variables (*targets*) given the value of certain other variables (*evidence*). The desired inference could either be in the predictive or in the diagnostic direction, or even a combination of the two.

Our proposed revision algorithm, based on existing theory-revision techniques (Ourston and Mooney, 1994; Opitz and Shavlik, 1993), makes use of this flow of reasoning. The data is assumed to be a set of examples consisting of observations of the evidence and the target variables. For each example, the revision algorithm first propagates the evidence to the target variables using the standard inference algorithms used with Bayesian networks. Whenever the qualitative likelihood of a target variable does not match the data, it is marked as an error. Using qualitative analysis, this error is propagated to the rest of the network to produce a set of candidate revisions that would correct the erroneous target. Some of these revisions are then implemented and the network is retrained to revise the parameters. These steps are repeated until further revision fails to improve the performance of the network on the data.

In designing the details of the algorithms, we are concerned with the following issues: *when* should a network be modified, *which* components of the network should be modified, *what* kinds of modifications should be attempted, how to pick the *best* revision from the candidate set of revisions, and when to *stop* revising the network.

From our perspective, a network should be revised when it fails to predict the targets accurately. In the binary case, if a target variable is inferred to be *positive* (i.e. it is more likely to be true than false), but is specified as being *negative* in the data, then the example is defined to be a *false positive*. A *false negative* is similarly defined. Starting at a variable classified as a false positive or a false negative, our algorithm will propagate blame and revisions based on rules derived from the qualitative relationships between the variables in the network. This propagation terminates at terminal and evidence nodes.

The high-level algorithm for revising the structure of the network is as follows:

- Train the initial network to revise the parameters.
- Repeat until desired training performance
  - For each misclassified training example
    1. Propagate structural revisions starting at the target variables (using the heuristics discussed below).
    2. For each of the structural revisions hypothesized, add the example to the set of examples covered by the revision.

– Find a small set of revisions that covers all the misclassified examples using a greedy set covering algorithm.

– Train the revised network to revise the parameters.

Our algorithm uses heuristics for proposing revisions. These heuristics suggest a set of variables that can be held responsible for causing an erroneous prediction. They also suggest ways of revising the network around these variables in order to correct the error. These heuristics, therefore, restrict the search space by focusing the revision on a subset of all the variables in the network. A variable that is a false positive can be corrected by increasing the proportion of negative evidence presented to it, either by increasing the negative evidence or by decreasing the positive evidence. The qualitative relationships between the variables provide a way of proposing revisions that would lead to such corrections. Thus the following revisions could be proposed to correct a *false negative noisy-or* node.

1. Classify a negative parent as *false negative* and recursively apply the revision algorithm to the parent.

2. Classify a negative child with diagnostic evidence as *false negative* and recursively apply the revision algorithm to the child.

3. Add a positive node to the parent set of the node.

4. Make a positive node, with diagnostic evidence, a new child of the node.

An added complexity in deriving these heuristics is that variables with a common child can interact when the child has diagnostic evidence. Depending on the *synergy* of this interaction, additional revisions can be hypothesized. For example, the following revisions can be applied to correct a *false negative* variable with a *noisy-or* child,

1. Delete the link from a positive spouse to the common child, provided the common child has positive diagnostic evidence.

2. Classify a positive spouse as a false positive and recursively apply the revision algorithm to the spouse.

These rules seem intuitive given a logical interpretation of the *noisy-or* gate and one might question the role of the QPN theory in our analysis. The advantage of basing our rules on the QPN idea is that similar rules can be derived for other kinds of specialised nodes as long as the qualitative relations associated with it are known.

Whenever a revision that adds a parent (or a child) to a node is implemented, a new intermediate node is created between the node and the new parent (child). For example, let us suppose that, for the network shown in Figure 12a, it is determined that the revision that the covers the most number of misclassified examples (*false negatives*) is that of adding a link between the variables E and C. Figure 12b shows the network resulting from the implementation of this revision. Notice that a new *noisy-and* variable $H1$ has been added as an intermediate node between variables E and C. It is possible that this revision, while accounting for a large number of *false negatives*, could give rise to *false positives*. In other words, adding a link between E and C could cause the network to over-generalise. These *false positives* would be detected during the next iteration of the training process. Suppose that, in the next iteration of the structure revision algorithm, it is determined that the best way to revise the theory, in order to correct these *false positives*, is to add a link between the nodes $H1$ and D. Figure 12c shows the network resulting from implementing this revision. Node D has been linked to $H1$ via a new, *noisy-or* node $H2$. At the end of training, all branches of the network that form a single chain can be collapsed into a single link. For instance, if further iterations of revision do not modify the portion of the network shown in the figure, then the single chain connecting variables $H1$, $H2$ and D can be collapsed to result in the network shown in Figure 12d.

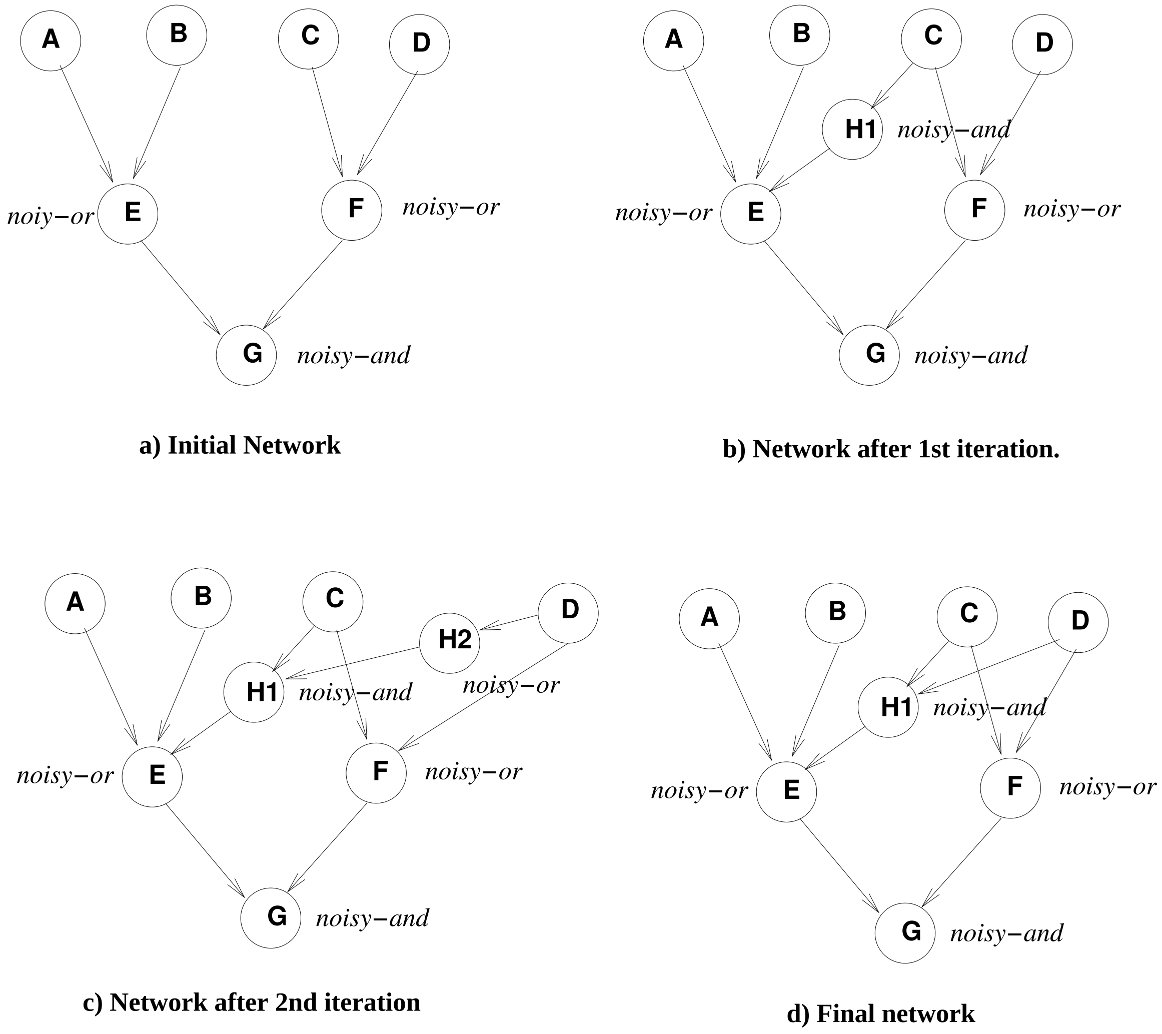

Figure 12: Adding Hidden Variables: An example

So far, we have presented a high-level view of the algorithm for proposing revisions to a given Bayesian network and for adding hidden variables. A number of important issues have yet to be addressed, and the details of each step in the algorithm have to fleshed out. The approach outlined here is similar to the one used by (Opitz and Shavlik, 1993) to revise knowledge-based neural networks. Their experiments revealed that adding links and new nodes is more crucial for their system than deleting links and nodes. This is because the weights on the extra links in the neural network eventually decay to insignificance during the parameter learning phase. We would like to run similar experiments to determine if a similar phenomenon can be observed in our system. Focusing the revision algorithm just on adding nodes and links would significantly reduce the search space.

5.3 Future Work

Working out the details of the structure revision algorithm will be the main focus of our research in the immediate future. Once the algorithm has been implemented, we will evaluate its performance on some real-world problems. For this purpose, we will try to find some real-world applications of Bayesian networks which could be used for evaluation. We will also compare the performance of our system with other systems, such as K2 (Cooper and Herskovits, 1992), that learn Bayesian networks, We discuss this in greater detail in Section 6.1.

6 Research Plan

As discussed in previous chapters, we have already implemented our technique for revising the conditional probabilities for predictive inference. Preliminary results show that our technique is effective in learning the parameters of a Bayesian network given the structure. We also have an outline of the algorithm for structure revision. Topping our research agenda is the task of working out the details of the structure revision algorithm. We also propose to evaluate our parameter revision system by comparing it with other such techniques and to explore further the idea of using non-gradient techniques for training our neural networks. We have discussed these ideas in detail in the previous sections. In this section, we will elaborate on our proposed empirical evaluation of our system. In addition, we will also discuss some extensions that we would like to explore as a part of this research.

6.1 Empirical Evaluation

We want to show, through experiments, that our technique can indeed revise networks so as to improve its performance on a given task. To this end, we will evaluate our system on some learning problems by testing how well the learned network can generalise to cover unseen cases. Next, we would also like to compare our system to some of the other systems that learn Bayesian networks. In particular, we would like to empirically prove the hypothesis that it is better to train a network to optimise it for its intended task rather than optimise the probability of the network given the data. For this, we will compare the performance of our system with the APN (Russell et al., 1995) approach as well as the EM algorithm. Both these systems only learn the parameters of the network and cannot revise the structure. Therefore, we will also compare the performance of our system with systems like K2 (Cooper and Herskovits, 1992) which use induction to learn the structure of a network. This experiment will also evaluate the advantage of using the theory-revision approach to learning as opposed to a purely inductive approach.

A standard practice in the field of learning Bayesian networks is to the evaluate a learning algorithm by comparing the learned network with the original network, called the *gold standard* (Heckerman, 1995). This assumes that the target network is known ahead of time. Given the target network, the practice is to generate data from the network, learn a Bayesian network from the generated data, and compare the learned network with the original. Our system requires an initial network to bias the learning. For this, we will perturb the original network randomly to create the initial network, which will then be revised using the generated data. One way of comparing the original and the learned networks is to compute the *cross-entropy*, which measures how close the learned distribution is to the target distribution (Heckerman et al., 1994). Another measure is the *structural-difference* measure, which computes how similar the structures of the two networks are. One of the data set that is used as a standard for such experiments is the ALARM data set, which is a database of cases generated from a network designed to assist in monitoring the heart rate of a patient (Beinlich et al., 1989; Cooper and Herskovits, 1992). Russell et al. (1995) evaluate their system on data generated from a network for car insurance risk estimation (Musick, 1994). We will evaluate our system on both these data sets. We will also test our system on a database of cases generated from a Bayesian network for a medical knowledge-base which is used in evaluating the ability medical students to treat patients (Pradhan et al., 1994).

Real world learning problems, however, seldom provide gold standards. While we will certainly evaluate our algorithm using the methodology described above, we also propose to evaluate it using real-world data for which approximate theories are available but not the target networks. The evaluation measure, in these cases, would be the prediction accuracy of the revised network on unseen data. Apart from using the *splice junction* data set, we will also evaluate our system for building a network to predict the aspect of a verb in a sentence (Marshall R. Mayberry, 1995).

6.2 Extension to general nodes

So far we have only considered networks with *noisy-or* and *noisy-and* nodes. It would be useful to extend our techniques to revise networks with general nodes. Schwalb (1993) proposes a technique for mapping a Bayesian network with general nodes into a neural network and learning the parameters using backpropagation techniques. It should be straightforward to incorporate his techniques into our parameter revision system. A more challenging task would be to extend our structure revision algorithm to cover general nodes. As we have pointed out earlier, so long as the qualitative relationship between the variables are known, our algorithm will be able to determine possible revisions, regardless of the actual type of nodes present in the network. However, in the case of general nodes, it is not always possible to isolate individual qualitative relationships, since the CPTs only specify the combined effect of all the parents of a node. These are some of the challenging issues that we hope to address in the future.

6.3 Extension to multi-valued variables

While all our algorithms have assumed that the variables in the network are binary-valued, several real-world applications of Bayesian networks require multi-valued variables. The notion of *noisy-or* gates has been extended to handle multi-valued variables (Pradhan et al., 1994). *Noisy-and* gates can be similarly extended. We believe that the parameter revision component can be extended to this case easily. Extending the theory revision algorithm poses a challenge, since we now have to search for hidden variables with an unknown number of values. This would increase the search space considerably. Connolly (1993) proposes a technique for discovering hidden variables with an unknown number of values using a clustering algorithm similar to COBWEB (Fisher, 1987). However, their approach can only learn tree-structured networks with all the observable variables at the bottom-most level. We will consider adapting such algorithms to our learning problem.

7 Related Work

The problem of learning Bayesian networks from data has received considerable attention in recent years. On one hand, several techniques have been developed that use statistical tests to determine the causal relationship among the observed variables (Pearl and Verma, 1991; Glymour and Spirtes, 1988). However, such methods require exhaustive search and are, as such, inefficient.

Cooper and Herskovits (1992) moved away from this paradigm and proposed a Bayesian approach to learning Bayesian networks. They viewed the problem as one of maximising the probability of the network given the data. They proposed a scoring metric that can be used to incrementally hill-climb through the space of networks to find one that is highly probable given the data. Others (Buntine, 1991; Heckerman et al., 1994; Provan and Singh, 1994) have followed up on this paradigm, introducing variations to improve the performance of the algorithm. The problem with these approaches, again, is that they involve extensive search and are not very efficient for discovering hidden variables.

Apart from algorithms for learning the structure of a network, several researchers have also studied the problem of learning the parameters of a network given the structure. This is fairly straightforward when all the variables of the network are represented in the data. A common approach is to use the maximum likelihood estimates for the parameters, which in the case of no hidden variables, reduces to a function of the relative frequencies of occurrences of the values of the variable.

In the case of data that is missing some values, approximation methods like *Gibbs Sampling* (Geman and Geman, 1984) and *EM* (Dempster et al., 1977; Lauritzen, 1995) have been proposed. Both these methods require some initialisation of the parameters and data for the missing variables. The complete data is then sampled to compute new values for the parameters. These steps are repeated until some convergence criteria is met.

While the above methods are efficient when the parameters have certain kinds of distributions, gradient descent approaches have been suggested for general distributions. One example is the idea of *Adaptive Probabilistic Networks* (APNs) (Russell et al., 1995), which uses a gradient descent algorithm to find the parameters that maximise the probability of the data given the network. This is different from our approach, which tries to find parameters that will maximise the predictive accuracy of the network. We believe that training a network to be optimal for the task it is being built to perform will result in better networks. However, APNs can be used with networks that have loops, whereas our technique only works with polytrees. We would like to point out that since our technique for theory revision is modular, with a clear separation between the structure revision and the parameter estimations components, we can use any of the other parameter estimation methods with our revision algorithm.

Musick (1994) uses statistical technique for induction of the parameters of a Bayesian network assuming that the structure is given. Each parameter is represented as a distributions rather than as a point value. The focus of this research is more on techniques for inferring with a Bayesian network with parameters specified as distributions rather than on the induction itself. He does address the question of inventing hidden variables, but the solution proposed works only for very limited cases.

One of the early connectionist approaches to learning the parameters of a Bayesian networks is that reported in Neal (1992). It also uses the *noisy-or* approximation of a Bayesian node. However, since it uses stochastic networks similar to the Boltzmann machine, simulation of the network involves allowing the network to settle down to an equilibrium for each pattern observed. This is expensive and slows down learning. We use a forward propagation algorithm which results in faster training.

Schwalb (1993) addresses the problem of learning the parameters of a given Bayesian network by mapping it onto a neural network with SIGMA-PI nodes and learning the conditional probabilities associated with the network (represented by link weights in the corresponding neural network) using standard backpropagation techniques (McClelland and Rumelhart, 1988). This has the advantage that it is able to learn the conditional probabilities even in the presence of hidden variables. However, the size of the neural network is combinatorial in the number of parents a node has in the corresponding Bayesian network, making the technique infeasible for even modestly large networks.

While the problem of inducing Bayesian networks from data has been explored deeply, the problem revising a Bayesian network has received very little attention. Buntine (1991) has proposed a technique for revising a Bayesian network efficiently, using scoring metrics similar to that proposed by (Cooper and Herskovits, 1992). However, he does not specify any method for recognising *when* the network needs to be revised. Nor does he discuss ways of focusing on the portions of the network that should be modified. As such, his approach involves extensive search and therefore is inefficient, especially in the presence of hidden variables.

Lam and Bacchus (1994) have a technique for incrementally refining a Bayesian network using the Minimum Description Length (Rissanen, 1978) principle. Their approach, however, can only modify those portions of the network whose variables are observable. Thus, it cannot modify nor invent a hidden variable.

Taking a different perspective, our proposed research also fits into the class of research that is concerned with combining connectionist and symbolic approaches to learning. Thus, our research is related to KBANN in that it uses backpropagation to refine an existing domain theory inductively. However, a clear advantage of our proposed system is that the network with its nodes and parameters has well defined semantics under the formalism of Bayesian networks.

Our research can also be viewed as an extension of RAPTURE, described in (Mahoney and Mooney, 1993a; Mahoney and Mooney, 1994). While RAPTURE is concerned with applying symbolic and connectionist techniques to revise certainty factor rule bases, we address the issue of doing the same with Bayesian networks.

Apart from Schwalb (1993), most of the research in learning Bayesian networks has taken the approach of optimising the probability of the network given the data. However, as mentioned previously, Schwalb (1993) is only concerned with learning the parameters of the network. Moreover, the technique presented handles only the predictive case. Our approach is unique in that it applies the idea of optimising the network for its intended task to the problem of learning the structure as well as the parameters of the network.

A survey of related research shows that, while the problem of learning Bayesian networks with no hidden variables is well-understood, learning networks with hidden variables is still an open problem. We hope to make a significant contribution in that direction with our proposed research.

A significant component of research that is lacking in the area of learning Bayesian networks is rigorous experimental comparisons of the various approaches to determine their strengths and weaknesses. One of the goals of this research is to provide such experimental comparisons. This will serve to not only evaluate our approach, but also to provide a clear view of what problems have been addressed in this area and what problems remain an issue.

Recently, there has been a lot of interest in *Bayesian backpropagation* (Buntine and Weigend, 1991; Buntine, 1994). We would like to point out that our research is very different in that it addresses a different problem. *Bayesian backpropagation* proposes a Bayesian theory of the back-propagation algorithm used for training standard feed-forward neural networks, whereas our research addresses the problem of learning the parameters of a Bayesian network using backpropagation techniques.

8 Conclusion

Bayesian networks provide an elegant and theoretically sound formalism for representing and reasoning with uncertainty. Given that knowledge acquisition in general is a recognised problem, it is not surprising that building knowledge bases in the form of Bayesian networks is difficult. However, the need to specify the conditional probability tables in precise numeric tables makes the problem of acquiring Bayesian networks even harder. Therefore, learning Bayesian networks from data is beginning to receive a lot of attention recently. Rapid progress has been made in solving some of the learning problems. However, some problems, such as learning a Bayesian network in the presence of hidden variables are still open.

The field of inductive learning provides a wealth of techniques for acquiring knowledge from data. Machine learning researchers have studied several representations, both symbolic and subsymbolic. Research in this field has also revealed that, for some learning problems, revising an initial approximate theory to fit the data produces more accurate results with lesser data than learning from data alone. However, not much has been done apply these techniques to the problem of learning Bayesian networks.

With all the recent activity in this area, the time seems ripe for bringing together the techniques developed in the area of symbolic and connectionist learning to this problem. In this proposal, we have explored one such synergy which uses ideas from symbolic and connectionist learning to address some of the weaknesses in the existing techniques for learning Bayesian networks. Our technique will make a significant contribution to this area by providing an efficient way to revise a network in the presence of hidden variables and imperfect structure.

We have implemented a part of our proposed technique and preliminary experiments have shown it to be effective. However, much remains to be done. We have yet to address, in detail, the problem of structure revision. We also have to carry out extensive experiments to evaluate the effectiveness of the overall system.

References

Beinlich, I., Suermondt, H., Chavez, R., and Cooper, G. (1989). The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, 247–256. London, England.

Buchanan, G., and Shortliffe, E., editors (1984). *Rule-Based Expert Systems:The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley Publishing Co.

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 52–60.

Buntine, W. L. (1994). Operations for learning graphical models. *Journal of Artificial Intelligence Research*, 2:159–225.

Buntine, W. L., and Weigend, A. S. (1991). Bayesian back-propagation. *Complex Systems*, 5(6):603–643.

Burnell, L., and Horovitz, E. (1995). Structure and chance: Melding logic and probability for software debugging. *Communications of the Association for Computing Machinery*, 38(3):31–41.

Cohen, W. (1992). Compiling prior knowledge into an explicit bias. In *Proceedings of the Ninth International Conference on Machine Learning*, 102–110. Aberdeen, Scotland.

Connolly, D. (1993). Constructing hidden variables in Bayesian networks via conceptual clustering. In *Proceedings of the Tenth International Conference on Machine Learning*, 65–72. Amherst, MA.

Cooper, G. (1990). Computational complexity of probabilistic inference using Bayesian belief networks (research note). *Artificial Intelligence*, 42:393–405.

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

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38.

Fahlman, S., and Lebiere, C. (1989). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems, Vol. 2*, 524–532. San Mateo, CA: Morgan Kaufmann.

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

Frean, M. (1990). The Upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209.

Fung, R., and Del Favero, B. (1995). Applying Bayesian networks to information retrieval. *Communications of the Association for Computing Machinery*, 38(3):42–48.

Geman, S., and Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 6:721–742.

Glymour, C., and Spirtes, P. (1988). Latent variables, causal models and overidentifying constraints. *Journal of Econometrics*, 39:175–198.

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052.

Heckerman, D., Geiger, D., and Chikering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 293–301. Seattle, WA.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison Wesley.

Lam, W., and Bacchus, F. (1994). Using new data to refine a Bayesian network. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 383–390.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201.

Mahoney, J. J., and Mooney, R. J. (1993a). Combining connectionist and symbolic learning to refine certainty-factor rule-bases. *Connection Science*, 5:339–364.

Mahoney, J. J., and Mooney, R. J. (1993b). Combining neural and symbolic learning to revise probabilistic rule bases. In Hanson, S., Cowan, J., and Giles, C., editors, *Advances in Neural Information Processing Systems, Vol. 5*, 107–114. San Mateo, CA: Morgan Kaufman.

Mahoney, J. J., and Mooney, R. J. (1994). Comparing methods for refining certainty-factor rule bases. In *Proceedings of the Eleventh International Conference on Machine Learning*, 173–180. New Brunswick, NJ.

Marshall R. Mayberry, I. (1995). A Bayesian approach to translating Japanese verb aspect to english. Report for a class project. The University of Texas at Austin.

McClelland, J. L., and Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. Cambridge, MA: The MIT Press.

McQuesten, P. (1995). Abductive Banner. Report for a class project. The University of Texas at Austin.

Mezard, M., and Nadal, J. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, A22(12):2191–2203.

Mozer, M. C., and Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems, Vol. 1*, 107–115. San Mateo, CA: Morgan Kaufmann.

Musick, R. C. (1994). *Belief Network Induction*. PhD thesis, University of California at Berkeley.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Noordewier, M. O., Towell, G. G., and Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, vol. 3. San Mateo, CA: Morgan Kaufman.

Opitz, D. W., and Shavlik, J. W. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 512–517. Chamberry, France.

Ourston, D., and Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 815–820. Detroit, MI.

Ourston, D., and Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:311–344.

Pazzani, M., and Kibler, D. (1992). The utility of background knowledge in inductive learning. *Machine Learning*, 9:57–94.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo,CA: Morgan Kaufmann, Inc.

Pearl, J., and Verma, T. (1991). A theory of inferred causation. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 441–452.

Pradhan, M., Provan, G., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 484–490. Seattle, WA.

Provan, G. M., and Singh, M. (1994). Learning Bayesian networks using feature selection. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Richards, B. L., and Mooney, R. J. (1995). Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19(2):95–131.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Russell, S., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1146–1152. Montreal, Canada.

Schwalb, E. (1993). Compiling Bayesian networks into neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, 291–297. Amherst, MA.

Shortliffe, E., and Buchanan, B. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379.

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

Srinivas, S., and Breese, J. (1993). Ideal: Influence diagram evaluation and analysis in lisp: Documentation and users' guide. Technical Report Technical Memorandum No. 23, Rockwell International Science Center, Palo Alto: Rockwell.

Thompson, K., Langley, P., and Iba, W. (1991). Using background knowledge in concept formation. In *Proceedings of the Eighth International Workshop on Machine Learning*, 554–558. Evanston, IL.

Towell, G. G., and Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 861–866. Boston, MA.

Wellman, M. P. (1990). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44:257–303.