Learning Belief Networks in the Presence of Missing Values and Hidden Variables

Nir Friedman
Computer Science Division, 387 Soda Hall
University of California, Berkeley, CA 94720
nir@cs.berkeley.edu

Abstract

In recent years there has been a flurry of works on learning probabilistic *belief networks*. Current state of the art methods have been shown to be successful for two learning scenarios: learning both network structure and parameters from *complete* data, and learning parameters for a fixed network from *incomplete* data—that is, in the presence of missing values or hidden variables. However, no method has yet been demonstrated to effectively learn network structure from incomplete data.

In this paper, we propose a new method for learning network structure from incomplete data. This method is based on an extension of the *Expectation-Maximization* (EM) algorithm for model selection problems that performs search for the best structure *inside* the EM procedure. We prove the convergence of this algorithm, and adapt it for learning belief networks. We then describe how to learn networks in two scenarios: when the data contains missing values, and in the presence of hidden variables. We provide experimental results that show the effectiveness of our procedure in both scenarios.

1 INTRODUCTION

Belief networks (BN) (also known as *Bayesian networks* and *directed probabilistic networks*) are a graphical representation for probability distributions. They are arguably the representation of choice for uncertainty in artificial intelligence. These networks provide a compact and natural representation, effective inference, and efficient learning. They have been successfully applied in expert systems, diagnostic engines, and optimal decision making systems (e.g., [Heckerman et al. 1995]).

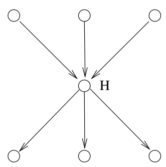
A Belief network consists of two components. The first is a directed acyclic graph in which each vertex corresponds to a random variable. This graph represents a set of conditional independence properties of the represented distribution. This component captures the *structure* of the probability distribution, and is exploited for efficient inference and decision making. Thus, while belief networks can represent arbitrary probability distributions, they provide computational advantage for those distributions that can be represented with a simple structure. The second component is a collection of *local interaction models* that describe the conditional probability of each variable given its parents in the graph. Together, these two components represent a unique probability distribution [Pearl 1988].

Eliciting belief networks from experts can be a laborious and expensive process in large applications. Thus, in recent years there has been a growing interest in learning belief networks from data [Cooper and Herskovits 1992; Lam and Bacchus 1994; Heckerman et al. 1995].[1] Current methods are successful at learning both the structure and parameters from *complete* data—that is, when each data record describes the values of all variables in the network. Unfortunately, things are different when the data is *incomplete*. Current learning methods are essentially limited to learning the parameters for a fixed network structure.

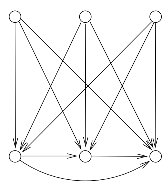
This is a significant problem for several reasons. First, most real-life data contains *missing values* (e.g., most of the non-synthetic datasets in the UC Irvine repository [Murphy and Aha 1995]). One of the cited advantages of belief networks (e.g., [Heckerman 1995, p. 1]) is that they allow for principled methods for reasoning with incomplete data. However, it is unreasonable at the same time to require complete data for training them.

Second, learning a concise structure is crucial both for avoiding overfitting and for efficient inference in the learned model. By introducing *hidden* variables that do not appear explicitly in the model we can often learn simpler models. A simple example, originally given by Binder et al. [1997], is shown in Figure 1. Current methods for learning hidden

[1] We refer the interested reader to the tutorial by Heckerman [1995] that overviews the current state of this field.



(a)



(b)

Figure 1: (a) An example of a network with a hidden variable. (b) The simplest network that can capture the same distribution without using the hidden variable.

variables require that human experts choose a fixed network structure or a small set of possible structures. While this is reasonable in some domains, it is clearly infeasible in general. Moreover, the motivation for learning models in the first place is to avoid such strong dependence on the expert.

In this paper, we propose a new method for learning structure from incomplete data that uses a variant of the *Expectation-Maximization* (EM) [Dempster et al. 1977] algorithm to facilitate efficient search over large number of candidate structures. Roughly speaking, we reduce the search problem to one in the complete data case, which can be solved efficiently. As we experimentally show, our method is capable of learning structure from non-trivial datasets. For example, our procedure is able to learn structures in a domain with several dozen variables and 30% missing values, and to learn the structure in the presence of several hidden variables. (We note that in both of these experiments, we did not rely on prior knowledge to reduce the number of candidate structures.) We believe that this is a crucial step toward making belief network induction applicable to real-life problems.

To convey the main idea of our approach, we must first review the source of the difficulties in learning belief networks from incomplete data. The common approach to learning belief networks is to introduce a scoring metric that evaluates each network with respect to the training data, and then to search for the best network (according to this metric). The current metrics are all based, to some extent, on the *likelihood* function, that is, the probability of the data given a candidate network.

When the data is complete, by using the independencies encoded in the network structure, we can decompose the likelihood function (and the score metric) into a product of terms, where each term depends only on the choice of parents for a particular variable and the relevant *statistics* in the data—that is, counts of the number of common occurrences for each possible assignment of values to the variable and its parents. This allows for a modular evaluation of a candidate network and of all local changes to it. Additionally, the evaluation of a particular change (e.g., adding an arc from X_1 to X_2) remains the same after changing a different part of the network (e.g., removing an arc from X_1 to X_3). Thus, after making one change, we do not need to reevaluate the score of most of the possible neighbors in the search space. These properties allow for efficient learning procedures.

When the data is incomplete, we can no longer decompose the likelihood function, and must perform *inference* to evaluate it. Moreover, to evaluate the optimal choice of parameters for a candidate network structure, we must perform non-linear optimization using either EM [Lauritzen 1995] or gradient descent [Binder et al. 1997]. In particular, the EM procedure iteratively improves its current choice of parameters Θ using the following two steps. In the first step, the current parameters are used for computing the *expected* value of all the statistics needed to evaluate the current structure. In the second step, we replace Θ by the parameters that maximize the complete data score with these expected statistics. This second step is essentially equivalent to learning from complete data, and thus, can be done efficiently. However, the first step requires us to compute the probabilities of several events for each instance in the training data. Thus, learning parameters with the EM procedure is significantly slower than learning parameters from complete data.

Finally, in the incomplete data case, a local change in one part of the network, can affect the evaluation of a change in another part of the network. Thus, the current proposed methods evaluate all the neighbors (e.g., networks that different by one or few local changes) of each candidate they visit. This requires many calls to the EM procedure before making a single change to the current candidate. To the best of our knowledge, such methods have been successfully applied only to problems where there are few choices to be made: Clustering methods (e.g., [Cheeseman et al. 1988; Chickering and Heckerman 1996]) select the number of values for a single hidden variable in networks with a fixed structure, and Heckerman [1995] describes an experiment with a single missing value and five observable variables.

The novel idea of our approach is to perform search for the best structure *inside* the EM procedure. Our procedure maintains a current network candidate, and at each iteration it attempts to find a better network structure by computing the expected statistics needed to evaluate alternative structures. Since this search is done in a complete data setting, we can exploit the properties of scoring metric for effective search. (In fact, we use exactly the same search procedures as in the complete data case.) In contrast to current practice, this procedure allows us to make a significant progress in the search in each EM iteration. As we show in our experimental validation, our procedure requires relatively few EM iterations to learn non-trivial networks.

The rest of the paper is organized as follows. We start in Section 2 with a review of learning belief networks. In Section 3, we describe a new variant of EM, the *model selection EM* (MS-EM) algorithm, and show that by choosing a model (e.g., network structure) that maximizes the complete data score in each MS-EM step we indeed improve the objective score. In Section 4, we discuss the details of applying the MS-EM algorithm for learning belief networks using the *minimal description length* scoring metric. In Section 5, we present experimental evaluation of this procedure both for handling missing values and learning structure with hidden nodes. Finally, in Section 6 we discuss the implications of our results and possible future extensions.

2 REVIEW OF LEARNING BELIEF NETWORKS

Consider a finite set $\mathbf{U} = \{X_1, \ldots, X_n\}$ of discrete random variables where each variable X_i may take on values from a finite set. We use capital letters, such as X, Y, Z, for variable names and lowercase letters x, y, z to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, and assignments of values to the variables in these sets are denoted by boldface lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$.

A *belief network* is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathbf{U}$. Formally, a belief network for $\mathbf{U}$ is a pair $B = \langle G, \Theta \rangle$. The first component, namely G, is a directed acyclic graph whose vertices correspond to the random variables $X_1, \ldots, X_n$ that encodes the following set of conditional independence assumptions: each variable X_i is independent of its non-descendants given its parents in G. The second component of the pair, namely Θ, represents the set of parameters that quantifies the network. It contains a parameter $\theta_{x_i|\Pi_{x_i}} = P(x_i|\Pi_{x_i})$ for each possible value x_i of X_i, and Π_{x_i} of Π_{X_i}, where Π_{X_i} denotes the set of parents of X_i in G. A belief network B defines a unique joint probability distribution over $\mathbf{U}$ given by:

$$P_B(X_1, \ldots, X_n) = \prod_{i=1}^{n} P_B(X_i|\Pi_{X_i})$$

The problem of learning a belief network can be stated as follows. Given a *training set* $D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ of instances of $\mathbf{U}$, find a network B that *best matches* D. The common approach to this problem is to introduce a scoring function that evaluates each network with respect to the training data, and then to search for the best network (according to this metric). The two main scoring functions commonly used to learn belief networks are the *belief scoring* function [Cooper and Herskovits 1992; Heckerman et al. 1995], and the one based on the principle of *minimal description length* (MDL) [Lam and Bacchus 1994] which is equivalent to Schwarz' *Bayesian information criterion* (BIC) [Schwarz 1978]. In this extended abstract we concentrate on the MDL/BIC metric. We defer treatment of the Bayesian metric to the full version of this paper.

Let $B = \langle G, \Theta \rangle$ be a belief network, and let $D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ be a training set where each $\mathbf{x}^i$ assigns a value to some (or all) variables in $\mathbf{U}$. The MDL score of a network B given a training data set D, written *Score*$(B : D)$, is given by the following equation:

$$\mathit{Score}(B : D) = L(B : D) - \frac{\log N}{2}\#(B) \qquad (1)$$

where $\#(B)$ is the number of parameters in the network.[2] The first term is the of the *log-likelihood* of B given D: $L(B : D) = \sum_{i=1}^{N} \log(P_B(\mathbf{x}^i))$. The log-likelihood has a statistical interpretation: the higher the log-likelihood is, the closer B is to modeling the probability distribution in the data D. The second term is a penalty term that biases the score metric to prefer simpler networks. (We refer the interested reader to [Heckerman 1995; Lam and Bacchus 1994] for detailed description of this score.)

When the all instances $\mathbf{x}^i$ in D are *complete*—that is, they assign values to all the variables in $\mathbf{U}$—the log-likelihood term decomposes. Let $N_{\mathbf{X}}(\mathbf{x})$ be *statistics* of $\mathbf{x}$ in D—that is, the number of instances in D where $\mathbf{X} = \mathbf{x}$. (Note that $N_{\mathbf{X}}(\cdot)$ is well-defined only for complete datasets.) We omit the subscript of $N_{\mathbf{X}}$ whenever it is clear from the context. Applying the definition of P_B to the log-likelihood and changing the order of summation yields the following well-known decomposition of the log-likelihood according to the structure of B: $L(B : D) = \sum_i \sum_{x_i, \Pi_{x_i}} N(x_i, \Pi_{x_i}) \log(\theta_{x_i|\Pi_{x_i}})$. It is easy to show that this expression is maximized when $\theta_{x_i|\Pi_{x_i}} = \frac{N(x_i, \Pi_{x_i})}{N(\Pi_{x_i})}$. Thus, given a network structure, there is a closed form solution for the parameters that maximize the log-likelihood score. Moreover, since the second term of (1) does not depend on the choice of parameters, this solution maximizes the MDL score.

Using this decomposition, we have that for each network structure B the score decomposes as well: *Score*$(B : D) = \sum_i \mathit{Score}_i(\Pi_{X_i}, \Theta_i : D)$, where

$$\begin{aligned}\mathit{Score}_i(\Pi_{X_i}, \Theta_i : D) \quad &= \sum_{x_i, \Pi_{x_i}} N(x_i, \Pi_{x_i}) \log(\theta_{x_i|\Pi_{x_i}}) \\ &\quad - \frac{\log N}{2}\#(X_i, \Pi_{X_i}) \qquad (2)\end{aligned}$$

and $\#(X_i, \Pi_{X_i})$ is the number of parameters need to represent $P(X_i \mid \Pi_{X_i})$. As explained in the introduction, this decomposition is crucial for learning structure. A *local* search procedure that changes one arc at each move can efficiently evaluate the gains made by adding or removing an arc. Such a procedure can also reuse computations made in previous stages to evaluate changes to the parents of all variables that have not been changed in the last move. One particular search procedure that exploits this decomposition is a greedy hill-climbing procedure that at each step

[2] The MDL scoring metric is often defined as the negative inverse of (1), where learning attempts to minimize the score rather than maximize it.

performs the local change that results in the maximal gain, until it reaches a local maxima. Although this procedure does not necessarily find a global maxima, it does perform well in practice; e.g., see [Heckerman et al. 1995].

When the training data is *incomplete*, that is, some $\mathbf{x}^i$s do not assign values to all the variables in $\mathbf{U}$, the situation is quite different. In this case the log-likelihood, and consequently the MDL score, do not decompose. Moreover, we cannot choose parameters in each local interaction model independently of the others. These issues have a drastic effect on the learning procedure. To evaluate the score of a structure, we must find the optimal parameter setting (see [Chickering and Heckerman 1996]). This usually involves some form of parametric search (e.g., gradient descent [Binder et al. 1997] or EM [Lauritzen 1995]). Moreover, for each candidate structure we consider, we must reevaluate the choice of parameters, since, in general, we cannot adopt the parameters computed for previous candidates.

3 THEORETICAL FOUNDATIONS

The standard EM algorithm is a method for parametric estimation for problems with missing data [Dempster et al. 1977; McLachlan and Krishnan 1997; Tanner 1993]. Here we present an extension of EM, which we call the *model selection EM* (MS-EM) algorithm, that deals with model selection as well as parameter estimation.

We start with some notation. Assume we have an input dataset with N records. We define random variables $\mathbf{U}^N = \{X_i^j : 1 \le i \le n, 1 \le j \le N\}$ such that X_i^j describes the assignment to X_i in the j'th input record. Let $\mathbf{O} \subseteq \mathbf{U}^N$ be the set of *observed* variables, that is, those whose values are specified in D, and let $\mathbf{H} = \mathbf{U}^N - \mathbf{O}$ be the *hidden* variables. We assume that we have a class of *models* $\mathcal{M}$ such that each model $M \in \mathcal{M}$ is parameterized by a vector Θ_M such that each (legal) choice of values Θ_M defines a probability distribution $P(\cdot : M, \Theta_M)$ over $\mathbf{U}^N$. From now on we use Θ as a shorthand for Θ_M when the model M is clear from the context.

We also assume that we want to find the choice of (M, Θ) that maximizes a scoring metric of the form:

$$S_O(M, \Theta) = \log P(\mathbf{O} : M, \Theta) - \mathit{Pen}(M, \Theta, O).$$

The term $\log P(\mathbf{O} : M, \Theta)$ is the *log-likelihood* of data given the choice of model, and term $\mathit{Pen}(M, \Theta, O)$ is some penalty function that might depend on the values of the observable variables in the training data. We assume that had we observed the complete data, we would have been able to maximize this score. Unfortunately, we do not have these values. However, we can examine *expected score* by taking expectation over all possible values $\mathbf{H}$ might take. To do so, we need to estimate the probability of these assignments. Given a particular estimate (M^*, Θ^*), the *expected score* is:

$$Q(M, \Theta : M^*, \Theta^*) = E[\log P(\mathbf{O}, \mathbf{h} : M, \Theta) - \mathit{Pen}(M, \Theta, O)],$$

where the expectation is over the value of $\mathbf{h}$ according to $P(\mathbf{h} \mid \mathbf{O} : M^*, \Theta^*)$.

The MS-EM algorithm can be now stated concisely:

Procedure MS-EM:
  Choose M^0 and Θ^0 randomly.
  Loop for $n = 0, 1, \ldots$ until convergence
    Find a model M^{n+1} that maximizes $Q(\cdot : M^n, \Theta^n)$
    Let $\Theta^{n+1} = \arg\max_\Theta Q(M^{n+1}, \Theta : M^n, \Theta^n)$

That is, at each stage we choose a model and parameters that have the highest expected score given our previous assessment. For any particular instantiation of this algorithm, we have to show that we can indeed find a pair (M, Θ) that maximizes the expected score. In fact, it is not necessary to maximize the expected score: it suffices that at each iteration we choose M^{n+1} and Θ^{n+1} so that $Q(M^{n+1}, \Theta^{n+1} : M^n, \Theta^n) > Q(M^n, \Theta^n : M^n, \Theta^n)$.

This process converges when there is no further improvement in the objective score. That is, when $S_O(M^{n+1}, \Theta^{n+1}) = S_O(M^n, \Theta^n)$. In practice, we stop the procedure when the change in the objective score is negligible (i.e., smaller than 0.5 percent.)

To show that this algorithm is useful, we need to show that it indeed improves the objective score in each iteration.

Theorem 3.1: *If $Q(M, \Theta : M^*, \Theta^*) > Q(M^*, \Theta^* : M^*, \Theta^*)$, then $S_O(M, \Theta) > S_O(M^*, \Theta^*)$*

Thus, if at each iteration we choose (M^{n+1}, Θ^{n+1}) that has a higher expected score than the previous candidate, we are bound to improve the objective score. The proof of this theorem is a relatively simple extension of the corresponding proof for parametric EM [McLachlan and Krishnan 1997].

This theorem shows that the MS-EM makes progress at each step until it converges. What do we know about the point of convergence this algorithm? Using results for standard EM, we have that if $Q(M^*, \Theta^* : M^*, \Theta^*) = \arg\max_\Theta Q(M^*, \Theta : M^*, \Theta^*)$, then Θ^* is a stationary point of $S_{O,M^*}(\Theta) = S_O(M^*, \Theta)$; that is, the gradient at that point is zero [McLachlan and Krishnan 1997]. This means that the choice of parameters at the point of convergence is either a local maxima, local minima, or a saddle point of S_{O,M^*}.

What can we say about the choice of model at the convergence point? The formal notion of stationarity does not apply to the discrete space $\mathcal{M}$ of possible models. However, we can define the class of "stationary points" of this algorithm as all the points to which the algorithm can converge. Note that this set of points is a subset of the set of stationary points in spaces of parameterization of all the candidate models in $\mathcal{M}$. Thus, had we run standard (e.g., parametric) EM for each of these, we would have potentially encountered at least the same number of stationary points.

This discussion suggests another modification in the learning algorithm. As we shall see below, maximizing the choice of parameters Θ for a fixed model is computationally cheaper than searching for a better model. Thus, we can modify our algorithm so that it alternates between iterations that optimize the parameters for the current model candidate, and iterations that search for a different model. We call this procedure the *alternating MS-EM* (AMS-EM):

procedure AMS-EM:
  Choose M^0 and $\Theta^{0,0}$ randomly.
  Loop for $n = 0, 1, \ldots$ until convergence
    Loop for $l = 0, 1, \ldots$ until convergence or $l = l_{\max}$
      Let $\Theta^{n,l+1} = \arg\max_\Theta Q(M^n, \Theta : M^n, \Theta^{n,l})$
    Find a model M^{n+1} that maximizes $Q(\cdot : M^n, \Theta^{n,l})$
    Let $\Theta^{n+1,0} = \arg\max_\Theta Q(M^{n+1}, \Theta : M^n, \Theta^{n,l})$

This variant attempts to make progress using parametric EM steps, for some specified number of steps, or until convergence. Then it considers making additional progress by changing the choice of model. This can lead to computational advantages, and moreover, as we shall see in Section 5.2, in some situations it can avoid early convergence to undesirable local maxima.

4 MS-EM FOR LEARNING BELIEF NETWORKS

To apply the MS-EM algorithm to learning belief networks with the MDL score, we need to show how to choose a model (G, Θ) that increases the expected score. As usual for models in the exponential family, we get that the expected score has the same form as the complete data score. Using the definition of the MDL score, and the linearity of expectation, we get that $Q(G, \Theta : G^*, \Theta^*) = \sum_i Q_i(\Pi_{X_i}, \Theta_i : B^*)$, where $B^* = (G^*, \Theta^*)$ and

$$Q_i(\Pi_{X_i}, \Theta_I : B^*) = \sum_{x_i, \Pi_{x_i}} E[N(x_i, \Pi_{x_i}) \mid \mathbf{O}] \log(\theta_{x_i|\Pi_{x_i}}) - \frac{\log N}{2} \#(X_i, \Pi_{X_i}),$$

where we take expectation according to P_{B^*} Thus, we get an analogue to the decomposition of the MDL score for the complete data case. The only difference is that we use the *expected statistics* based on the model B^* instead of actual statistics. As in the complete data case, we maximize the expected score, for a particular network structure G, by setting $\theta_{x_i|\Pi_{x_i}} = \frac{E[N(x_i,\Pi_{x_i})|\mathbf{O}]}{E[N(\Pi_{x_i})|\mathbf{O}]}$. As a consequence, we can use the same search strategies that exist for the complete data case.

The MS-EM algorithm for belief networks is implemented using the architecture described in Figure 2. The Search Engine module is responsible for choosing candidate networks to evaluate. For each candidate, it calls the Score

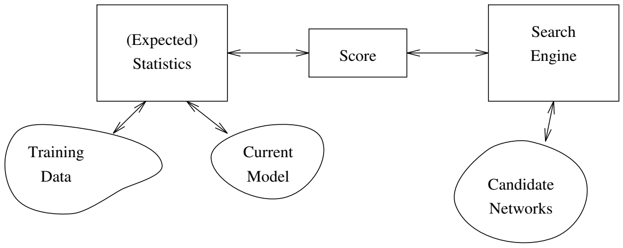


Figure 2: Architecture of the implementation of the MS-EM algorithm.

module, which implements the particulars of the scoring metric (e.g., MDL or Bayesian scoring). To evaluate the score, the Score module requires the corresponding statistics, which are supplied by the Statistics module. In the complete data case, the Statistics module answer queries by counting instances in the training data. In MS-EM, this module computes expected statistics, using a the network found in the previous iteration. Indeed, our implementation is based on a previous implementation of a complete-data learning software. The implemention of MS-EM involved minor changes to the Statistics module and an implementation of an inference algorithm.

Most of the running time during the execution our procedure is spent in the computations of expected statistics. This is where our procedure differs from parametric EM. In parametric EM, we know in advance which expected statistics are required. Namely, these are all statistics of events of the form X_i, Π_{X_i} where Π_{X_i} are the parents of X_i in the fixed network structure we are considering. Since these variables are also the parents of X_i in our current candidate, we can employ efficient inference algorithms that compute all the required statistics in one pass over the training data (e.g., the clique-tree algorithm of Lauritzen and Spiegelhalter [1988]).

In our procedure, we cannot determine in advance which statistics will be required. Thus, we have to handle each query separately. Moreover, when we consider different structures, we are bound to evaluate a larger number of queries than parametric EM . Thus, a "structural" iteration, where we search for a better network structures, is more expensive than a "parametric" iteration, where we only update the parameters. Since we usually need several iterations to converge on the best parameters, this suggests that by using the AMS-EM algorithm, we can save computational resources.

Additionally, we implemented two straightforward mechanisms to reduce the overhead in computing statistics. First, at the beginning of each iteration, we use the clique-tree algorithm to compute the statistics for the current candidate. This is the only computation needed during the parametric optimization steps in AMS-EM, and these statistics are also used in initial phases of the search in MS-EM. Second, the

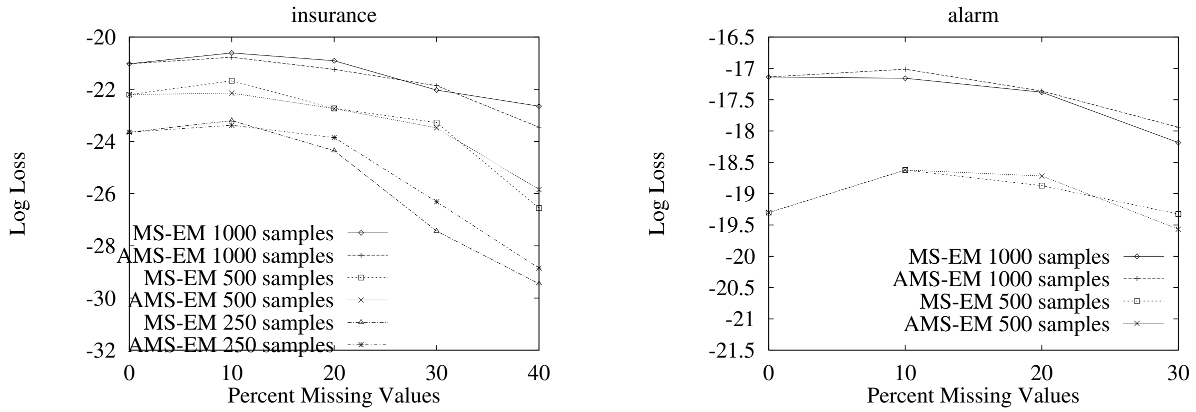


Figure 3: Plots showing the degradation in learning performance as a function of the percentage of missing values. The horizontal axis shows the percentage of missing values, and the vertical show the log-loss (higher is better).

Statistics module caches queries computed using the current network. This allow us to avoid computing a query (or marginals of it) more than once at each iteration.

Another crucial point is the choice of the initial model for MS-EM. Clearly. this choice determines the convergence point of the algorithm (since the algorithm itself is deterministic). In general, we do not want to choose too simple initial structure, since such a structure embodies many independencies, and thus biases the expected statistics to indicate that variables are independent of each other. On the other hand, we do not want to choose too complex initial structure, since such a structure might be too hard to perform inference on. In the following sections, we discuss particular choices of initial structure in the scenarios we consider. As usual in EM procedures, we choose the initial parameters for the chosen structure randomly. Moreover, in order to avoid unfortunate early convergence, we usually run the procedure several times, starting with different initial points. We then choose the network with the highest score from those found by these runs.

5 EXPERIMENTAL RESULTS

In this section we report initial experimental results for two scenarios of incomplete data: missing values and hidden variables. We discuss aspects of the procedure that are relevant to each type of problem, and how we address them. In both settings, we evaluate our procedure as a *density estimator*, that is, how well it learns the distribution in the training data. Another interesting aspect we measure for belief networks is how well they model the structure of the domain (e.g., number of correct arcs found). We defer this analysis to the full version of this paper.

5.1 Missing Values

Many real life data sets contain missing values. In order to evaluate our procedure, we performed the following experiment that examines the degradation in performance of our learning procedure as a function of the percentage of missing values.

In this experiment we generated artificial training data from two networks: **Insurance**—a network for classifying car insurance applications [Binder et al. 1997] that has 26 variables; and **alarm**—a network for intensive care patient monitoring [Beinlich et al. 1989] that has 37 variables. From each network we randomly sampled training sets of different sizes. We then randomly removed values from each of these training sets to get training sets with varying percentage of missing values. For each data point we generated five independent training sets.

For each training set, we started the procedure from five random initial points. We choose the initial structure to be a random chain-like network that connected all the variables. We suspect that for most missing value problems with relatively low number of missing values (e.g., less than $10-15\%$), the topology of the initial network is not crucial. We hope to verify this experimentally in the future. In both experiments, we tried both the MS-EM procedure and the AMS-EM procedure.

We evaluated the performance of the learned networks by measuring how well they model the target distribution. To do so, we sampled a test set of 10,000 instances and evaluated the average *log-loss* of each learned network on this test set, that is, $\frac{1}{N}\sum_{i=1}^{N}\log P_B(\mathbf{x}^i)$.[3] (We used the same test set for evaluating all the learned networks.) The results are summarized in Figure 3. As expected, there is a degradation in performance when there are large number of missing

[3]This is a standard method for evaluating density estimates.

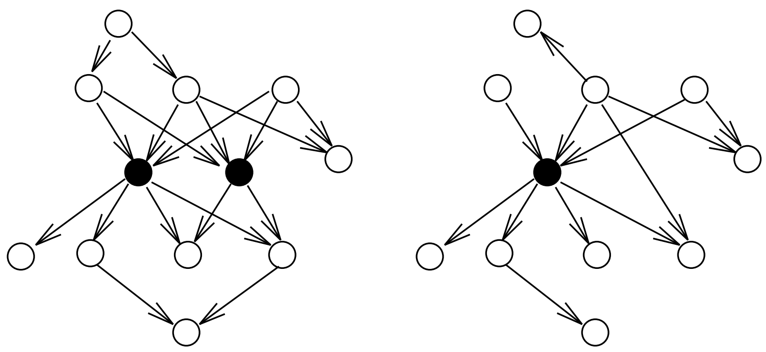


Figure 4: (a) The network **3x2x4** used in the experiments. The shaded nodes correspond to hidden variables. (b) One of networks learned from 1000 samples using one hidden variable.

variables. However, for reasonable percentage of missing values, the degradation is moderate or non-existent.[4] These results are encouraging, since they show that even with 30% of the values missing, our procedure usually performs better than one that receives half the number of complete instances. (Note that with 30% missing values, virtually all the instances in the dataset are incomplete). These results also show that the AMS-EM algorithm roughly performs as well as the MS-EM algorithm.

We note that in this experiment, values where missing at random. In many real-life domains, this is not the case. In these domains, the pattern of missing values can provide additional information (e.g., the patient was too ill to take a certain test). Such information can be easily modeled by introducing for each X_i a new variable labeled "Seen-X_i" that denotes whether X_i was observed or not. Datasets annotated with this information can then be handled by our procedure. This allows the learning procedure to learn correlations between the observations of various variables and value of other variables. This approaches utilizes the belief network to describe both the underlaying domain and the process of observation in the domain. We hope to return to this issue in future work.

5.2 Hidden Variables

In most domains, the observable variables describe only some of the relevant aspects of the world. This can have adverse effect on our learning procedure. Consider, for example, a medical domain in which the training data consists of observed symptoms (e.g., fever, headache, blood pressure etc.), and the medication prescribed by the doctor. One *hidden* quantity that we not observe is which disease(s) the patient has. Knowledge of the patient's disease makes the treatment independent of most of the symptoms. On the other hand, when we do not observe the disease, all observables seem related to each other. Thus, we hope that by introducing hidden variables, we will be able to learn simpler models that are less prone for overfitting and more efficient for inference.

Thus, there is growing interest in learning networks that include one or more hidden variables. Unfortunately, the only methods that learn with hidden variables must restrict themselves to small number of structure candidates (as done by Cheeseman et al. [1988] and Chickering and Heckerman [1996]) since the cost of running EM to find the parameter setting for each candidate structure is high. Our procedure allows us to learn network structures with hidden variables. The following experiments attempt to show the effectiveness of the procedure.

Before we describe the experiments, we explain how we choose the initial network our procedure. We start with two simple observations. First, if a hidden variable H is isolated in the network we use for computing expected statistics, then the expected statistics would show that it is independent of all other variables. Thus, the MS-EM algorithm would not add arcs to or from H. Second, suppose that we learn a network where H is a leaf (e.g., there are no arcs leading out of H) or H is a root with only one child. In this case, marginalizing H from the learned distribution does not effect the distribution over the observable variables. Thus, H does not contribute to the representation of the domain.

These observations suggest that a hidden variable is beneficial only if it is connected to other variables in the networks. Moreover, once we lose that property, we are effectively ignoring the hidden variable from there on. Thus, we want to ensure that our choice of initial structure connects hidden variables to several observable variables. One structure that ensures that this is the case is a bipartite graph in which all of the hidden variables are parents of each observable vari-

[4] We suspect that the slight improvement in the score for 10% is due to the randomness of our procedure. This apparently allows the search to escape some of the local maxima found when there are no missing values.

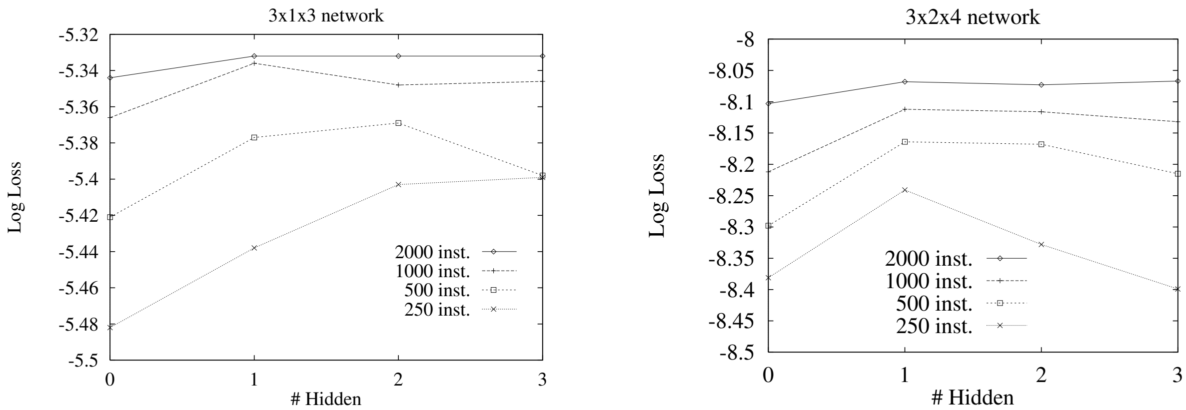


Figure 5: Plots showing learning performance when learning models with hidden variables as a function of the number of hidden variables. The results shown are average loss from 5 different experiments in each sample size.

able. Of course, when there are many hidden variables, this network might require many parameters. In these cases, we randomly choose edges from the bipartite graph ensuring that each observable variable does not have more then a certain fixed number of parents.

As usual, we randomly choose parameters for this network. These random choices create weak dependencies between the observable variables and the hidden variables. However, if we immediately apply MS-EM with such an initial guess, we end up choosing a structure where most of the hidden variables are independent of the rest of the networks. To avoid this problem, we run several iterations of the standard EM procedure on initial network. These iterations change the parameters so that there are stronger dependencies between the hidden variables and the observables. After this preprocessing stage, we continue on as before (i.e., either MS-EM or AMS-EM). In our experiments there was no significant difference between MS-EM and AMS-EM, and thus, we focus on the latter in the reminder of this section.

In our experiment, we created a two networks: **3x1x3** with the topology shown in Figure 1a (where all the variables are binary); and **3x2x4** with the topology shown in Figure 4a Like the first network, this second network also has hidden variables "meditating" between two groups of observed variables. It includes, however, other variables that make the problem harder. We quantified both networks using randomly chosen parameters. We then sampled, from each network, five training sets of sizes 250, 500, 1000, and 2000 instances of the observable variables, and learned networks in the presence of 0, 1, 2, or 3 hidden binary variables using the AMS-EM algorithm. We tested the average log-loss of our procedure on a separate test set. The results are summarized in Figure 5. Experiments with the MS-EM algorithm led to similar results, which we omit here because of space limitations.

These results show that introducing hidden variables results in improved density estimation. However, when the number of instances grows larger, we are able to "afford" to learn more complex structures, and then the impact of adding hidden variables diminishes. Also note that for small samples additional hidden variables can lead to worst performance. We suspect that this is due to overfitting, and we are currently exploring this phenomena.

6 DISCUSSION

In this paper, we introduced a new method for learning belief networks from incomplete data. As the preliminary experimental results show, this method allows us to learn in non-trivial domains. We believe that these results will have impact in several domains. In particular, we are planning to explore application of this method for learning temporal models that involve hidden entities. Such models are crucial for applications in speech recognition, computational biology and reinforcement learning. Additionally, we are exploring the use of this method for classification tasks.

The main computational efforts in our procedure is computation of expected statistics during "structural" iterations, where the procedure performs search. Our current inference procedure is unoptimized and this has restricted the scope of the reported experiments. We are currently replacing this module by an optimized inference engine. Additionally, we are examining a *stochastic* variant of MS-EM (analogous to stochastic EM [McLachlan and Krishnan 1997; Tanner 1993]). In this variant, we use sampling procedures to complete that training data. That is, from each partial instance, we sample several complete instances (using our current estimate of the model). We then learn from the completed data using the complete data methods. This approach is appealing since unlike exact inference methods, the running time of sampling is not sensitive to the complexity of the network. Finally, we are examining alternative search procedures that attempt to escape local maxima by using various local perturbations.

An additional topic that deserves more attention is learning models with hidden variables. Our current procedure starts with a given set of hidden variables and attempts to find a model that includes them. A more sophisticated approach would create hidden variables on as-needed basis during the learning process. This requires making a decision as to when to add a new variable, and how to insert it into the current model. We are currently exploring several methods for recognizing where to insert a hidden variable.

Finally, in the current presentation we focused on the MDL learning score. An analogous development can be carried for the Bayesian learning score [Cooper and Herskovits 1992; Heckerman et al. 1995] in a relatively straightforward manner. Due to space restrictions, we defer the presentation of these issues to the full version of this paper.

Acknowledgments

I am grateful to Kevin Murphy, Stuart Russell, Geoff Zweig, and particularly Moises Goldszmidt and Daphne Koller for comments on earlier version of this paper and useful discussions relating to this topic. This research was supported by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341.

References

Beinlich, I., G. Suermondt, R. Chavez, and G. Cooper (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proc. 2'nd European Conf. on AI and Medicine*. Berlin: Springer-Verlag.

Binder, J., D. Koller, S. Russell, and K. Kanazawa (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning this volume*.

Cheeseman, P., J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman (1988). Autoclass: a Bayesian classification system. In *ML '88*.

Chickering, D. M. and D. Heckerman (1996). Efficient approximations for the marginal likelihood of incomplete data given a Bayesian network. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pp. 158–168.

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

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

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

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

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

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

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

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

McLachlan, G. J. and T. Krishnan (1997). *The EM Algorithm and Extensions*. Wiley Interscience.

Murphy, P. M. and D. W. Aha (1995). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. San Francisco, Calif.: Morgan Kaufmann.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics 6*, 461–464.

Tanner, M. A. (1993). *Tools for Statistical Inference*. New York: Springer-Verlag.