Efficient Learning using Constrained Sufficient Statistics

Nir Friedman
Institute of Computer Science
The Hebrew University
Ross building, Givat Ram
Jerusalem 91904 ISRAEL
nir@cs.huji.ac.il

Lise Getoor
Computer Science Department
Gates 1A-126
Stanford University
Stanford, CA 94305-9010
getoor@cs.stanford.edu

Abstract

Learning Bayesian networks is a central problem for pattern recognition, density estimation and classification. In this paper, we propose a new method for speeding up the computational process of learning Bayesian network *structure*. This approach uses constraints imposed by the statistics already collected from the data to guide the learning algorithm. This allows us to reduce the number of statistics collected during learning and thus speed up the learning time. We show that our method is capable of learning structure from data more efficiently than traditional approaches. Our technique is of particular importance when the size of the datasets is large or when learning from incomplete data. The basic technique that we introduce is general and can be used to improve learning performance in many settings where sufficient statistics must be computed. In addition, our technique may be useful for alternate search strategies such as branch and bound algorithms.

1 Introduction

In recent years there has been a growing interest in learning Bayesian networks from data (Cooper & Herskovits 1992, Lam & Bacchus 1994, Heckerman 1995, Heckerman, Geiger & Chickering 1995). The common approach to this problem 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 standard methods use heuristic search, such as greedy hill-climbing, to maximize the network score.

In this paper, we propose a new approach to learning structure from data that uses bounds to make the search more efficient. Our approach uses constraints imposed by the statistics already calculated to guide the search algorithm. This allows us to avoid collecting statistics that are not crucial for learning and thus save computation time. Our technique is of particular importance when the size of the database is large or when the entries in the database may have missing values. This new method is one step in making the learning of Bayesian networks tractable in real world settings.

Informally, suppose we are considering adding a parent Y to a node X. In order to compute the network score, we need to collect the statistics for X's new family, which consists of X, Y and X's current set of parents. This is done by passing through the data set and counting the occurrences of particular instantiations of the variables. Before making this pass, we have some information about X, namely the statistics we have already collected about X and its parents. Most often, we also have statistics for X and Y. We can use these to constrain the values of the counts for the new family. In particular, if the best score we can hope for relative to the constraints is not promising, we can focus our efforts on another potential search step, rather than investing the effort to compute an exact value for the network with this additional edge.

As we show experimentally, our method is capable of learning structure from data more efficiently than standard methods. We see a significant improvement in the score versus time performance profile: given a fixed amount of time, the network induced by our algorithm has a better score than the network produced by the standard learning algorithms. This savings is due to a decrease in the number of passes over the data set required for each step in the search process.

The rest of the paper is organized as follows. We start in Section 2 with a review of learning Bayesian networks. In Section 3, we analyze the constraints imposed by the statistics that we have collected from our database, and show how these constraints can be used to minimize the additional statistics that we must compute. In Section 4, we describe a variant of the structure learning algorithm that makes use of the bounds that we have computed and analyze its performance. We conclude with a summary and discuss future research directions.

2 Learning Bayesian Networks

Consider a finite set $\mathcal{X} = \{X_1, \ldots, X_n\}$ 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 $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$, and assignments of values to the variables in these sets are denoted by boldface lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$. Finally, let P be a joint probability distribution over the variables in $\boldsymbol{U}$, and let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be subsets of $\boldsymbol{U}$. The sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ are *conditionally independent* given $\boldsymbol{Z}$ if for all $\mathbf{x} \in \mathrm{Val}(\boldsymbol{X}), \mathbf{y} \in \mathrm{Val}(\boldsymbol{Y}), \mathbf{z} \in \mathrm{Val}(\boldsymbol{Z})$, $P(\mathbf{x} \mid \mathbf{z}, \mathbf{y}) = P(\mathbf{x} \mid \mathbf{z})$ whenever $P(\mathbf{y}, \mathbf{z}) > 0$.

A *Bayesian network* is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathcal{X}$. Formally, a Bayesian network for $\mathcal{X}$ 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 $X_1, \ldots, X_n$. The graph 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, Θ, represents the set of parameters that quantifies the network. It contains a parameter $\theta_{x_i|\pi_i} = P(x_i|\pi_i)$ for each possible value x_i of X_i, and π_i of Π_i. Here Π_i denotes the set of parents of X_i in G and π_i is a particular instantiation of the parents. A Bayesian network B specifies a unique joint probability distribution over $\mathcal{X}$ given by:

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

The problem of learning a Bayesian network can be stated as follows. Consider a finite set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of discrete random variables where each variable X_i may take on values from a finite set. Given a *training set* $D = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ of instances of $\mathcal{X}$, 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 scoring functions most commonly used to learn Bayesian networks are the *Bayesian scoring* metrics (Cooper & Herskovits 1992, Heckerman et al. 1995), and one based on the principle of *minimal description length* (MDL) (Bouckaert 1995, Lam & Bacchus 1994). Let G be the first component of a Bayesian network, namely a directed acyclic graph, and let D be a training set. We denote the score of G by $\mathrm{Score}(G : D)$. The MDL scoring function is given by the following equation:

$$\mathrm{Score}_{MDL}(G : D) = \max_{\Theta} \mathrm{L}(\langle G, \Theta \rangle : D) - \frac{\log N}{2} \#(G) \quad (1)$$

The first term is the *log-likelihood* of $B = \langle G, \Theta \rangle$ given D:

$$\mathrm{L}(B : D) = \sum_{\ell=1}^{N} \log(P_B(\mathbf{x}^\ell)).$$

The log-likelihood has a statistical interpretation: the higher the log-likelihood is, the more probable D is generated according to B. The second term is a penalty term that biases the score metric to prefer simpler networks, where N is the number of random variables in the network and $\#(G)$ is the number of parameters in the network. (We refer the interested reader to (Heckerman 1995, Lam & Bacchus 1994) for detailed description of this score.)[1]

The Bayesian score is derived using Bayesian reasoning. Without providing details on the derivation, this score is defined as:

$$\mathrm{Score}_{Bayes}(G : D) = \log P(G \mid D) = \log P(D \mid G) + \log P(G) - C$$

where C is a constant that does not depend on G and $P(D \mid G)$ is the integration over all possible parameter assignments to G,

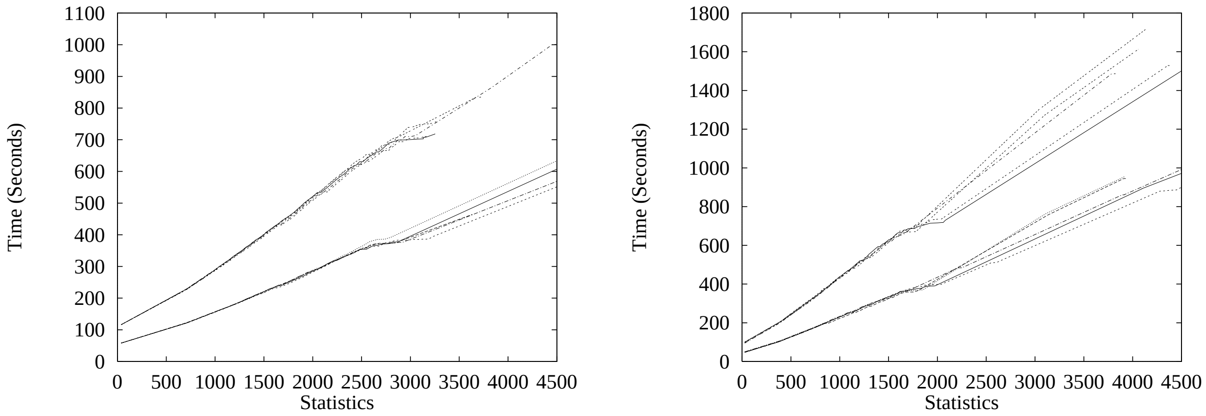
$$P(D \mid G) = \int P(D \mid G, \Theta) P(\Theta \mid G) d\Theta.$$

The particular choice of priors $P(G)$ and $P(\Theta \mid G)$ for each G determine the exact Bayesian score.

When all instances $\mathbf{x}^\ell$ in D are *complete*—that is, they assign values to all the variables in $\mathcal{X}$— the standard scoring functions are decomposable. That is, the scoring functions have the following general form: $\sum_i \mathrm{Score}(X_i \mid \Pi_i : N_{X_i,\Pi_i})$ where N_{X_i,Π_i} are the *statistics* of the variables X_i and Π_i in D—i.e., the number of instances in D that match each possible instantiation x_i and π_i. The variables X_i and Π_i are called the family of X_i.

By simple algebraic manipulation it is easy to show that, in the presence of complete data, the MDL score

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



(a) (b)

Figure 1: A plot of the number of sufficient statistics collected against running time of the learning procedure for runs of greedy hill-climbing search for 5 datasets of size 50,000 and 100,000 instances. (a) results for the "alarm" dataset, and (b) results for the "insurance" dataset.

decomposes in this way, and that each term depends only on the "local" statistics. In particular the log-likelihood term decomposes into terms that depend on the sufficient statistics of each family:

$$\text{Score}_{MDL}(G:D) = \sum_i \text{Score}_{MDL}(X_i \mid \Pi_i : N_{X_i,\Pi_i}),$$

where $\text{Score}_{MDL}(X_i \mid \Pi_i : N_{X_i,\Pi_i})$ is:

$$\max_{\theta_{X_i|\Pi_i}} \sum_{x_i,\Pi_i} N_{x_i,\pi_i} \log(\theta_{x_i|\pi_i}) - \frac{\log N}{2} \#(X_i \mid \Pi_i)$$

and $\#(X_i \mid \Pi_i)$ is the number of parameters need to represent $P(X_i \mid \Pi_i)$. It is straightforward to show that the choice of $\theta_{x_i|\pi_i}$ that maximizes this equation is $\theta_{x_i|\pi_i} = \frac{N_{x_i,\pi_i}}{N_{\pi_i}}$. We then have:

$$\text{Score}_{MDL}(X_i \mid \Pi_i : N_{X_i,\Pi_i}) = \sum_{x_i,\Pi_i} N_{x_i,\pi_i} \log(\frac{N_{x_i,\pi_i}}{N_{\pi_i}}) - \frac{\log N}{2} \#(X_i \mid \Pi_i)$$

Similarly, in the presence of complete data, if we restrict ourselves to certain class of *factorized* priors, such as the *BDe priors* of Heckerman et al. (1995), then the Bayesian score also decomposes:

$$\text{Score}_{BDe}(G:D) \quad = \quad \sum_i \text{Score}_{BDe}(X_i \mid \Pi_i : N_{X_i,\Pi_i}),$$

where $\text{Score}_{BDe}(X_i \mid \Pi_i : N_{X_i,\Pi_i})$ is:

$$\log \prod_{\pi_i} \frac{\Gamma(N'_{\pi_i})}{\Gamma(N'_{\pi_i} + \sum_{x_i} N_{x_i,\pi_i})} + \log \prod_{x_i} \frac{\Gamma(N'_{x_i,\pi_i} + N_{x_i,\pi_i})}{\Gamma(N'_{x_i,\pi_i})}$$

The counts $N'_{\pi_i} = N' \cdot P_{B'}(\pi_i)$ and $N'_{x_i,\pi_i} = N' \cdot P_{B'}(x_i,\pi_i)$ represent the prior assessment of the event using a prior network B' with weight N'.

This decomposition of the scores 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. An example of such a procedure is a greedy hill-climbing procedure that at each step performs the local change that results in the maximal gain, until it reaches a local maximum. Although this procedure does not necessarily find a global maximum, it does perform well in practice; e.g., see Heckerman et al. (1995). Although we focus on hill-climbing here, other local search procedures such as beam-search and simulated annealing can benefit from the methods we describe below.

Any implementation involves caching of computed counts to avoid unnecessary passes over the data. This cache also allows us to *marginalize* counts. Thus, if $N_{X,Y}$ is in the cache, we can compute N_X by summing over values of Y. This is usually much more efficient than making a new pass over the data. One of the dominating factors in the computational cost of learning from complete data is the number of passes actually made over the training data. This is particularly true when learning from very large training sets. The dependency of running time on sufficient statistics is illustrated Figure 1. This figure shows plot of time versus number of sufficient statistics for several runs on two datasets with varying sample sizes. We can see there is roughly a linear correlation between the running time and the number of statistics collected.

When the data is incomplete the situation is more complex. Without going into details, the problem is that the scores no longer decompose, and each candidate structure has to be evaluated using expensive parameter learning methods, such as EM. One method of learning structure in this situation extends EM to structure search. In the *Structural EM* procedure (Friedman 1997, Friedman 1998), we use the "current" candidate to compute *expected* counts from the data. Roughly speaking, the procedure computes the expected value of N_X, given the conditional distribution over missing values given the observed values and the current estimate of the distribution. We then learn a new structure as though these counts came from observable data. These steps are repeated, each iteration using the structure found in the previous one for computing expected counts, until convergence. The main theorems of Friedman (1997, 1998) show that this will always improve the MDL score, and indicate that it usually improves the BDe score. Thus, within structural EM, we can use the same search procedures we devise for complete data. The only difference is that counts are now the result of performing computations over the training instances, and thus, the computational cost of getting a new count is much more expensive than in the complete data case.

3 Using Score Bounds to Improve Search Performance

The greedy hill-climbing algorithm for learning Bayesian network structure maintains a current network hypothesis. At each step local modifications to the network, such as adding, deleting or reversing an edge, are considered. As noted above, we can compute the improvement that each such change will make by looking only at the local sufficient statistics in the neighborhood of that change.

We would like to use heuristic search, which is at the core of much of AI research, to guide our algorithm. When our heuristic information is of some minimal quality, and the information is not too expensive to compute, we can gain great benefits from using a more informed search method. Here we investigate the use of both guaranteed upper bounds and heuristic bounds.

A local search procedure examines small changes to the current candidate structure. To evaluate these changes, the procedure needs to examine the statistics of the new families in consideration. Intuitively, however, we do not need to seriously consider all possible modifications during search; most of these modifications are not the optimal changes (in the sense of greedy hill climbing), and some, if not most, actually do not improve the score. The question is how to avoid spending time evaluating such unhelpful modifications.

For concreteness, suppose we are considering adding a parent Y to a node X (see Figure 2(a)). In order to evaluate the new score of the network, we need to compute the local score $\text{Score}(X|\boldsymbol{Z}, Y : N_{X,Z,Y})$, where $\boldsymbol{Z}$ is the current parents of X. To compute this term we need the statistics $N_{X,Z,Y}$. What can we say about this modification *before* we go out and collect these statistics? For one, if we can *bound* from above the score of this change, and if we see that there is a better option currently available to us, we can skip the computation of statistics for this change (at least for now). In the next few sections we consider how to find such upper bounds, based on our knowledge of other statistics from the same database, and then, in Section 4 we discuss exactly how to use such bounds within a search procedure.

3.1 Characterization of the Local Score

To bound the score, we need to find a value such that the score given any of the possible values for $N_{X,Y,Z}$ consistent with such constraints will be smaller. A rough upper bound for the MDL score was suggest by Suzuki (1996):

$$\begin{aligned}
&\text{Score}_{MDL}(X_i \mid \Pi_i : N_{X_i,\Pi_i}) \\
&\quad = \sum_{x_i,\Pi_i} N_{x_i,\Pi_i} \log(\theta_{x_i|\Pi_i}) - \frac{\log N}{2} \#(X_i \mid \Pi_i) \\
&\quad \leq -\frac{\log N}{2} \#(X_i \mid \Pi_i).
\end{aligned}$$

This bound is based on the simple fact that the first term of the bound is non-positive (to see this, note that this term is the negative of the empirical *conditional entropy* $H(X_i \mid \Pi_i)$, which is non-negative (Cover & Thomas 1991)). Such a bound, however, totally ignores our (partial) knowledge about the data, and hence must be quite loose. A somewhat tighter bound can be found by considering properties of the conditional entropy.

In contrast to these loose bounds, we aim to find the *tightest* possible bound. This is done by maximizing the local scoring metrics from Section 2,

$$\max_{N_{X,Y,Z} \in \mathcal{C}} \text{Score}(X|Y, Z : N_{X,Y,Z})$$

where $\mathcal{C}$ is the feasible region of counts according to our current knowledge. Here the $N_{X,Y,Z}$ are the unknowns that we are trying to find, subject to the constraints imposed by the counts we have already computed, $N_{X,Y}$ and $N_{X,Z}$. Figure 2(b) shows a geometric view, where our unknowns form a three-dimensional grid, and we see our constraints as requirements placed on the row and column sums of the grid.

The key idea is that even before we collect the statistics $N_{X,Y,Z}$, we know quite a bit about the dataset. Since we have an edge from each parent in Z to X, presumably we have the statistics $N_{X,Z}$ in our cache. Similarly, we may have considered adding the edge $Y \to X$ at some point in our search, so we probably also have $N_{X,Y}$ in our cache. Certainly we can use these to constrain the possible values of $N_{X,Y,Z}$ in the dataset. Such constraints are simple linear constraints. For example, if we know $N_{X,Y}$, we can impose the constraint:

$$\sum_k N_{x_i,y_j,z_k} - N_{x_i,y_j} = 0,\ \forall i,j \qquad (2)$$

Similar linear constraints are implied by other partial statistics about X, Y, and Z.

Note that when learning from complete data, we know that the counts must be integers. Thus, we may want to focus on the integer solutions within the feasible region $\mathcal{C}$. However, the problem is simpler when we consider all points within the region, including fractional counts. (We also note that within the Structural EM algorithm we are dealing with expected counts which are usually not integer valued. Thus, in that case we need to consider all points within the region.) Let X be a three dimensional array of unknowns, where each $\mathsf{x}_{ijk} \in \Re$. Each x_{ijk} is in one-to-one correspondence with our discrete $N_{X,Y,Z}$ (each index i, j, k ranges over the indices of the instantiations of X,Y,and Z respectively).

We are interested only in the portion of the Score$_{MDL}$ that depends on the values of the counts. This is the log-likelihood term. We define the following objective function

$$F(\mathsf{X}) = \sum_{i,j,k} \mathsf{x}_{ijk} \log\left(\frac{\mathsf{x}_{ijk}}{\sum_{i'} \mathsf{x}_{i'jk}}\right)$$

We are interested in maximizing $F(\mathsf{X})$, subject to positivity constraints on x_{ijk} and the constraints implied by the counts we know. Thus, the constraint of (2) would be stated as:

$$\left(\sum_k \mathsf{x}_{ijk}\right) - N_{x_i,y_j} \quad = \quad 0,\ \forall i,j$$

We define $\mathcal{C}$ to be the feasible region defined by these constraints. It is easy to see that $\mathcal{C}$ is a convex set.

We begin by showing that, for strictly positive x_{ijk}, $F(\mathsf{X})$ is convex.

Lemma 3.1: *The function*

$$F(\mathsf{X}) = \sum_{i,j,k} \mathsf{x}_{ijk} \log\left(\frac{\mathsf{x}_{ijk}}{\sum_{i'} \mathsf{x}_{i'jk}}\right)$$

is convex over the positive quadrant, that is when x_{ijk} are all strictly positive.

Proof: (sketch) We show first that F is separable into the sum of independent functions, and then show that each of these individual terms is convex. Then, since the sum of convex functions on a convex region is also convex, F is convex.

The first step is straightforward.

$$\begin{aligned} F(\mathsf{X}) &= \sum_{j,k} \left(\sum_i \mathsf{x}_{ijk} \log\left(\frac{\mathsf{x}_{ijk}}{\sum_{i'} \mathsf{x}_{i'jk}}\right) \right) \\ &= \sum_{j,k} F_{jk}(\mathsf{X}_{\cdot jk}) \end{aligned}$$

where

$$F_{jk}(\mathsf{X}_{\cdot jk}) = \sum_i \mathsf{x}_{ijk} \log\left(\frac{\mathsf{x}_{ijk}}{\sum_{i'} \mathsf{x}_{i'jk}}\right)$$

Define $\mathsf{Y} = \mathsf{X}_{\cdot jk}$, $\mathsf{y}_i = \mathsf{x}_{ijk}$, and $F' = F_{jk}$:

$$\begin{aligned} F'(\mathsf{Y}) &= \sum_i \mathsf{y}_i \log\left(\frac{\mathsf{y}_i}{\sum_{i'} \mathsf{y}_{i'}}\right) \\ &= \sum_i \mathsf{y}_i \cdot \left(\log(\mathsf{y}_i) - \log\left(\sum_{i'} \mathsf{y}_{i'}\right) \right) \\ &= \sum_i \mathsf{y}_i \log(\mathsf{y}_i) - \left(\sum_i \mathsf{y}_i\right) \cdot \log\left(\sum_i \mathsf{y}_i\right) \end{aligned}$$

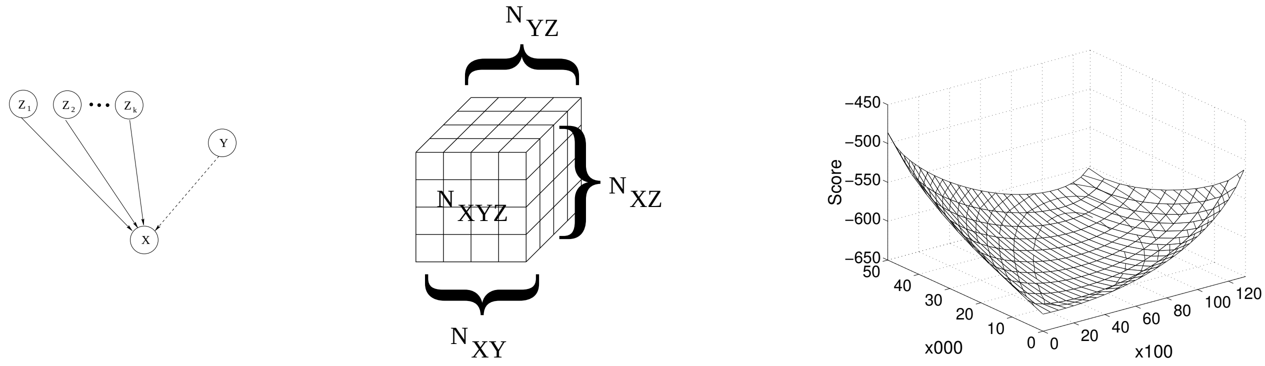
The Hessian, $\nabla^2 F'$, is:

$$\nabla^2 F' = \begin{cases} \frac{1}{\mathsf{y}_i} - \frac{1}{\sum_{i'} \mathsf{y}_i} & \text{if } i = j \\ -\frac{1}{\sum_{i'} \mathsf{y}_i} & \text{if } i \neq j \end{cases}$$

It is possible to show that, if each $\mathsf{y}_i > 0$, the Hessian is semi-positive definite. This implies F' is convex. Therefore, F is also convex. [2] ∎

Lemma 3.2: *The global maximum of the objective function F is achieved at an extreme point of the feasible region $\mathcal{C}$.*

[2] Intuitively, a function is convex if it is shaped like a bowl. If a function is strictly convex, the Hessian is positive definite. While F' is convex, it is not strictly convex, thus we may have a bowl with a flat bottom.



(a) (b) (c)

Figure 2: a) Adding an edge to the network b) A geometric view of the problem c) The score on the feasible region for a simple $2 \times 2 \times 2$ example. x_{000} and x_{100} are the two independent counts in the problem and score is shown on the z-axis.

Proof: As noted earlier, $\mathcal{C}$ is a bounded convex set. It is easy to see that the maximum of a convex function F is achievable at the boundary of the feasible region $\mathcal{C}$. Furthermore, the maximum is achievable at an *extreme point* of the feasible region. This follows from a straightforward argument, see for example (Luenbeger 1984). ∎

Theorem 3.3 : *The global maximum of $Score_{MDL}(X|Y,Z : N_{XYZ})$ is bounded by the global maximum of F and is achieved at an extreme point of the feasible region $\mathcal{C}$.*

Proof: Follows directly from the definition of F and the previous lemma. ∎

This result makes intuitive sense. Remember that we are asking which counts consistent with our current knowledge would make adding an edge most favorable. The fact that these counts must be extreme, implies that the local probability model at that node is as deterministic as possible. This is reasonable, since the more extreme the data (or the lower entropy of the data), the better is the score of a probabilistic network.

Although we limit our attention here to the MDL score, we note that the behavior of the BDe score is similar for sufficiently large counts. More precisely, our hypothesis is that if we bound the counts $N'_{x_i,y_i,z_i} + \mathsf{x}_{ijk}$ above some constant, then the BDe local score is also convex. When the counts can be smaller, the score is not well-behaved as x_{ijk} approaches 0. The upshot of this is that the situation for the BDe is a bit more complex, but techniques similar to those we discuss below would generate good approximate bounds also for this score.

3.2 Computing Bounds on the Local Score

Now that we have characterized the decision problem, we can use solutions to the problem in one of two ways. An assignment of values that satisfies our constraints is a *feasible solution* in the optimization terminology. A solution may be a local optimum, a global optimum or neither. We can use an optimal solution to bound the local score or we can use any feasible solution as a heuristic to guide our search.

How difficult is the task of finding the maximal value of F on $\mathcal{C}$? This problem seems at first similar to *entropy maximization.* In that problem, we attempt to find a probability distribution that maximizes the entropy function given some constraints. In entropy maximization the objective function is similar to the negative of our objective function (except for the penalty function). Thus, in our notation, these problems attempt to *minimize* F, and that can be done in a relatively straightforward way since F is convex—we only need to find the "bottom" of the bowl, and this can be done using, for example, gradient-based methods. Our optimization problem is to find the highest "vertex" of the bowl. The number of extreme points can be exponential in the number of constraints, and many of these can be local maxima. For example, in the example of Figure 2(c), there are two local maxima, only one of which is also the global maximum.

While we have reduced our search for a global optimum to extreme points of the feasible region, a method that enumerates the extreme points is clearly impractical, even for very small problems. Alternatively, we may consider a simplex style algorithm, that moves from extreme point to neighboring extreme points until no local improvement is possible. Such an algorithm is certainly guaranteed to find a local optimum. How-

			0	83
			369	~~417~~ 48
			0	158
			0	342
133	367	131	~~369~~ 0	

(a)

83	0	0	0	83
48	0	0	369	417
2	25	131	0	158
0	342	0	0	342
133	367	131	369	

(b)

0	0	0	83	83
133	0	131	153	417
0	138	0	0	158
0	209	0	133	342
133	367	131	369	

(c)

Figure 3: (a) Assigning a value using the MAXMAX heuristic (b) A complete feasible assignment for a slice using the MAXMAX heuristic (c) An example optimal assignment for the slice from an instance of a larger problem.

ever, since our objective function is nonlinear, it is not guaranteed to achieve a global optimum. Instead, if we wish to find a global optimum, we must resort to a nonlinear optimization procedure. There are many categories of nonlinear optimization algorithms. Here, we use a quasi-Newton method on the Lagrangian defined by our objective function and constraints. An attractive feature of this method is that it can be shown to converge superlinearly when initiated sufficiently close to the solution to the Lagrangian.

An alternative to finding the optimal solution is to consider *some* extreme point solution. While we cannot use them to bound our search, we can use them as a heuristic guide. How do we find an extreme point of our problem? It turns out that the constraints imposed by our cached counts have a particularly nice form that allow us to quickly find basic feasible solutions to our problem. We have counts $N_{X,Y}$ and $N_{X,Z}$. As we saw earlier, we can look at a particular "slice" of the problem for a specified value of X. The constraints describe an assignment problem, where we can view the $N_{x_i,Z}$ as row sums and view the $N_{x_i,Y}$ as column sums. The decision variables are the x_{ijk}. We can find a feasible solution using the following simple scheme:

1. Pick a row sum j with value r
2. Pick a column sum k with value c
3. Let $v = \min(r, c)$. Assign $\mathsf{x}_{ijk} = v$.
4. Update the column sum and row sum by subtracting v from each of them. Note that this will zero one of the two. This will force all of the unassigned entries in that row or column to be zero. See Figure 3(a).
5. Repeat until no more assignments can be made.

Each step of this process will zero at least one of the rows or columns, thus we make at most $n + m - 1$ assignments, where n is the number of rows and m is the number of columns. See Figure 3(b) for an example of a feasible assignment. Figure 3(c) shows an optimal assignment.

At this point, we can view the cost of each x_{ijk} assignment as $\mathsf{x}_{ijk} \cdot \log(\frac{\mathsf{x}_{ijk}}{\sum_{i'} \mathsf{x}_{i'jk}})$ and our problem is to find the minimum cost assignment. If these costs were a simple linear function of the x_{ijk}, then we could find the optimal solution by finding "augmenting" paths, which are analogous to searching along the extreme points of the simplex in the simplex algorithm. While we can find augmenting paths in our problem, as noted earlier, we would only be guaranteed to find a local maximum using such an algorithm.

Instead, we consider a simple heuristic for choosing i and j in each iteration of our algorithm for finding feasible solutions. At each point, we choose the row and column with the maximum row sum and the maximum column sum.[3] We call this the MAXMAX heuristic. The intuition here is that we would like to "pump-up" F as much as possible. We can do this by making some x_{ijk} as large as possible.

Table 4 compares the solutions found by the quasi-Newton method and the MAXMAX heuristics. In addition, we show another heuristic, RANDOM, that examines several random extreme points, and returns the best solution among them. We show the algorithms' performance for different sizes of randomly generated problems. Using the QUASI-NEWTON solution as a measure of the optimal solution, we see that the MAXMAX heuristic is in fact giving us some useful information about our bound. While these numbers give some flavor for the behavior of the different algorithms, in practice, our input will be anything but random. In then next section, we examine our algorithm's performance on more realistic input.

[3]We would like to thank Walter Murray for suggesting this heuristic.

Size $p \cdot m \cdot n$	MAXMAX	RANDOM	QUASI-NEWTON
2 2 2	-398.49	-463.19	-321.58
3 3 3	-766.78	-806.17	-647.94
4 4 4	-1250.33	-1290.38	-1062.35
5 5 5	-1752.09	-1852.65	-1548.67

Figure 4: The bounds heuristics give for randomly generated problems of various sizes. Here p, m and n are the dimensions of X, Y and Z respectively. The scores are unnormalized. Each score is the average of 1000 runs. MAXMAX uses a simple heuristic for finding a feasible solution. RANDOM examines 100 randomly generated extreme point solutions. QUASI-NEWTON uses a general nonlinear programming algorithm to find an optimal solution.

4 Experimental Results

We compared two search procedures. The first is the simple greedy hill-climbing procedure which we denote **HC**. In each iteration this procedure evaluates all possible modifications to the current candidate and incorporates the change that leads to the best improvement. Upon reaching a local maxima, the procedure applies a number of random modifications and restarts the search.

The second search procedure uses our current knowledge of the statistics to guide the search. We call this procedure *Constrained Sufficient Statistics* hill-climbing search, denoted **CSS-HC**. **CSS-HC** is essentially the same as **HC**—the only difference is in the evaluation of the best next move. This evaluation uses bounds in the following way. We start by computing the score of all the changes that can be evaluated using statistics in the cache. We then compute upper bounds on the scores of the remaining changes. At this stage we prune all the changes that are dominated by another modification. That is, the score of the pruned change or the upper bound on its score is smaller than the score of the other modification. If more than one change remains, we evaluate the true score of the change with the highest upper bound. We repeat this until only one modification remains undominated, and return it. It is easy to see that if we have true upper bounds, then this procedure returns the best possible change.

We evaluated these two search procedures on two domains. We generated training data from two networks: **alarm**—a network for intensive care patient monitoring (Beinlich, Suermondt, Chavez & Cooper 1989) that has 37 variables; and **Insurance**—a network for classifying car insurance applications (Binder, Koller, Russell & Kanazawa 1997) that has 26 variables. From each network we sampled training sets of sizes $10,000$, $50,000$ and $100,000$. We then plotted the *performance profiles* of both search procedures on these training sets. These profiles show the improvement in score as a function of either computation time or number of sufficient statistics computed. (This number roughly correlates with the number of passes over the training set—some statistics are computed by *marginalization* of statistics in the cache.) Figure 5 shows examples of such profiles. As we can clearly see, the **CSS-HC** procedures quickly zeros in on high scoring networks. Moreover, although we are not finding true upper bounds, **CSS-HC** converges to networks of the same quality as the standard **HC** procedure.

As we can see from Figure 5(a) and (b), the profile of score versus time and score versus the number of sufficient statistics computed are similar. This is consistent with the claim that the computation of sufficient statistics is indeed the determining factor in the running time of each algorithm. Finally, Figure 5(d) shows the scaling of the various algorithms with the size of the training data. As we can expect, the difference in performance grows when the cost of computation is higher.

In the results reported above, **CSS-HC** uses the MAXMAX heuristic to compute the bounds at each step. We also explored using the tighter bounds produced by a quasi-Newton algorithm. To calculate these bounds, we used an implementation in the NAG numerical software library for MATLAB. Interestingly, we found that the better bounds did not make a significant impact on our algorithm's performance. While computing more accurate bounds did adversely affect the running time, we did not see a measurable benefit in the number of statistics computed. This is an indication that our simple heuristic is in fact quite powerful: it is both cheap to compute and provides significant guidance.

5 Discussion

In this paper, we investigated methods for finding bounds on local scores based on partial knowledge about the dataset and showed how to use these bounds for speeding up Bayesian network learning. We are

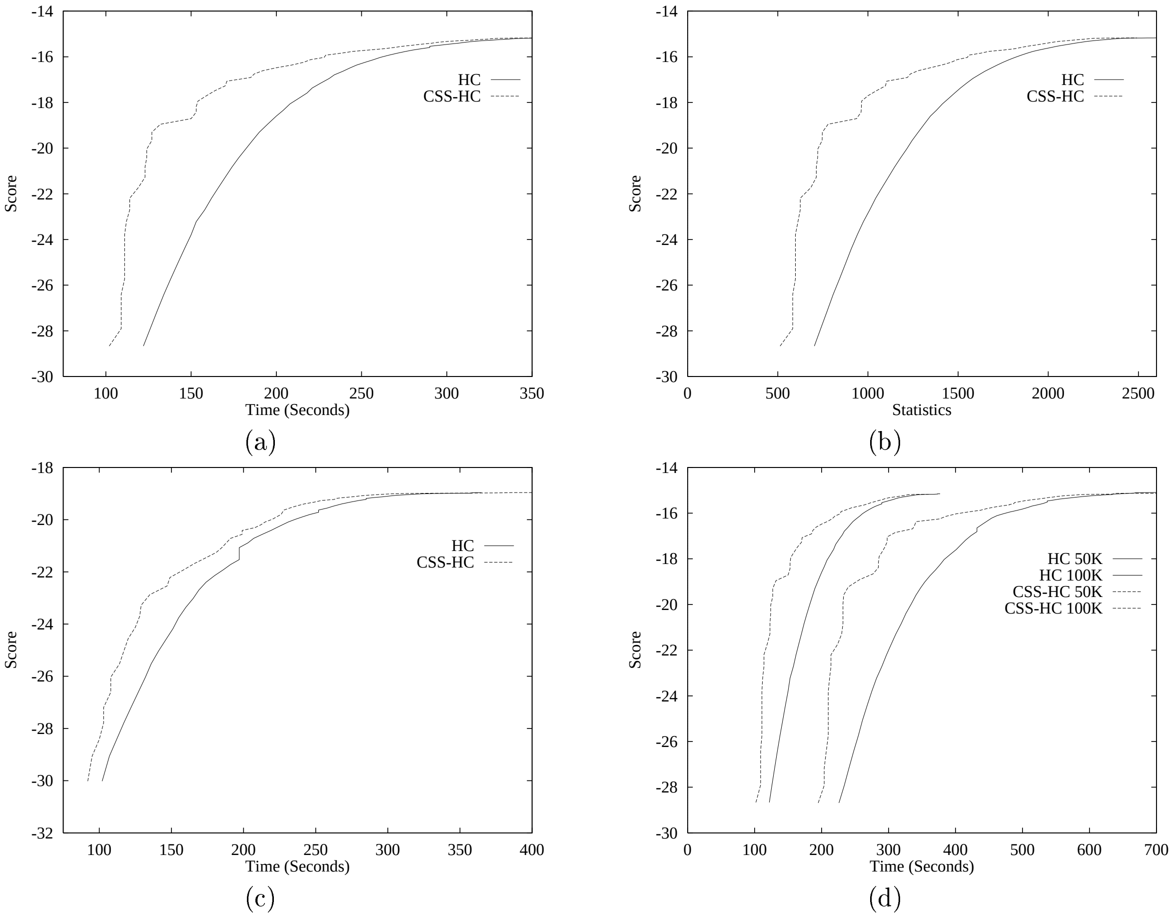


Figure 5: Performance profiles showing improvement in score versus either time or number of computed statistics: (a) and (b) Alarm domain with 50,000 instances. (c) Insurance dataset with 50,000 instances. (d) Performance for different dataset sizes on Alarm domain. The x-axis reports either time in units of seconds, or the number of computed statistics in the cache. The y-axis reports *normalized* scores, i.e., the score divided by the number of training instances.

currently working on a more extensive empirical investigation of these methods for learning from large real world datasets and in the presence of missing data (i.e., in the approach described in Friedman (1997, 1998)).

There are several dimensions for further research. Currently, we use the bounds within the overarching paradigm of greedy hill-climbing search. A promising avenue of research is using the bounds in a more global manner within the heuristic search. In particular, we are interested in developing a branch and bound search procedure for learning structure.

We stress that the general approach for bounding scores based on constraints on the possible sufficient statistics is not specific to Bayesian network learning. The same idea can be exploited in other settings such as decision tree learning and mining association rules. In addition, this approach is orthogonal to other methods that handle sufficient statistics efficiently, such as Moore & Lee (1997).

Acknowledgments

We are grateful to Walter Murray for his guidance on nonlinear optimization, and optimization software packages. In addition we would like to thank him for suggestions on heuristics for finding feasible solutions. We would like to thank Ronald Getoor for his help in showing that the Hessian of a certain class of matrices is positive semidefinite. We would also like to thank Peter Grunwald, Ron Parr, Daphne Koller and the members of the DAGS research group at Stanford for helpful comments. Most of the experiments reported here were run on the NOW cluster at UC Berkeley. We thank the NOW group for allowing us to use their resources. Some of this work was done while Nir Friedman was at University of California at Berkeley and was funded by ARO under grant number DAAH04-96-1-0341 and by ONR under grant number N00014-97-1-0941. Lise Getoor was supported by a National Physical Sciences Consortium fellowship. This work was also supported through the generosity of the Powell Foundation, by ONR grant N00014-96-1-0718, and ONR grant N66001-97-C-8554.

References

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

Binder, J., Koller, D., Russell, S. & Kanazawa, K. (1997), 'Adaptive probabilistic networks with hidden variables', *Machine Learning* **29**, 213–244.

Bouckaert, R. (1995), Bayesian Belief Networks: From Construction to Inference., PhD thesis, Utrecht University, Utrecht, The Netherlands.

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

Cover, T. M. & Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley & Sons, New York.

Friedman, N. (1997), Learning belief networks in the presence of missing values and hidden variables, *in* D. Fisher, ed., 'Proceedings of the Fourteenth International Conference on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 125–133.

Friedman, N. (1998), The Bayesian structural EM algorithm, *in* G. F. Cooper & S. Moral, eds., 'Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)', Morgan Kaufmann, San Francisco, CA.

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

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

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

Luenbeger, D. (1984), *Linear and Nonlinear Programming*, Addison Wesley, Reading, MA.

Moore, A. W. & Lee, M. S. (1997), 'Cached sufficient statistics for efficient machine learning with large datasets', *Journal of A.I. Research* **8**, 67–91.

Suzuki, J. (1996), Learning Bayesian belief networks based on the minimum description length principle: An efficient algorithm using the B & B technique, *in* L. Saitta, ed., 'Proceedings of the Thirteenth International Conference on Machine Learning', Morgan Kaufmann, San Francisco, CA, 462–470.