Understanding the effects of search constraints on structure learning

Michael Hay, Andrew Fast, and David Jensen
{mhay,afast,jensen}@cs.umass.edu
University of Massachusetts Amherst
Computer Science Department
Technical Report No. 07-21

April, 2007

Abstract

Recently, Tsamardinos et al. [2006] presented an algorithm for Bayesian network structure learning that outperforms many state-of-the-art algorithms in terms of efficiency, structure similarity and likelihood. The Max-Min Hill Climbing algorithm is a hybrid of constraint-based and search-and-score techniques, using greedy hill climbing to search a constrained space of possible network structures. The constraints correspond to assertions of conditional independence that must hold in the network from which the data were sampled. One would expect that constraining the space would make search both faster and more accurate, focusing search on the "right" part of the space. The published results indicate, however, that the resulting structures are less accurate when search is constrained. We reproduce these results and explain why they occur. At small samples, the statistical test of conditional independence has low power, which causes the algorithm to exclude edges between dependent variables. Also, the constraints make search relatively harder, leading to errors in edge orientation. In an unconstrained space, search can "repair" these errors by adding in extra edges. We conclude by proposing and evaluating an improved algorithm.

1 Introduction

There are two broad classes of Bayesian network structure learning algorithms: search-and-score methods seek a structure that best fits the given training data; constraint-based techniques try to infer the structure of the generating Bayesian network through tests of conditional independence. The two classes of algorithms use different criteria for selecting a structure. Constraint-based algorithms can be thought of as *structure matching*: they rule out structures that assert conditional independence relations that are not supported by data. In contrast, search-and-score techniques are *distribution approximating*: scoring structures by their fit to the data, penalizing for unwarranted complexity.

Recently, Tsamardinos et al. [2006] presented a hybrid algorithm that attempts to satisfy both of these objectives. The algorithm, called Max-Min Hill Climbing (MMHC), outperforms many state-of-the-art algorithms along three measures: efficiency, likelihood and structure similarity. The published results show, however, that unconstrained greedy search finds structures with higher likelihood. This is surprising, as the intuition behind the algorithm is that the search is focused only on the candidate structures that are consistent with conditional independence relations observed in the training data.

We observe that there are two main causes for the lower performance of the MMHC algorithm. At small samples, its statistical test of conditional independence has low power, which causes the algorithm to exclude edges between variables that are in fact dependent (type II errors). Second, while the constrained search space that MMHC considers does contain good structures, the constraints make it relatively harder to search than the unconstrained space. Specifically, the scoring metric used to evaluate structures during learning cannot distinguish the orientation of the edge, leading to errors. Unconstrained search can "repair" these errors by adding in extra edges; in the constrained space, such edges are forbidden by the constraints.

Using experimental analysis on several real-world networks from a diverse set of domains, we provide evidence supporting the above conclusions and eliminate several other plausible explanations. We outline some

possible improvements to the MMHC algorithm that could allow it to achieve the likelihood performance of Greedy Search while retaining its efficiency, and provide experimental evidence that one of the improvements works well. We conclude with a discussion of implications for Bayesian network structure learning.

2 Background

A Bayesian network structure is compact encoding of all of the conditional independence relations among a set of variables Pearl [1998]. The structure of the network is a directed acyclic graph G whose vertices correspond to the variables. Two variables X_i and X_j are conditionally independent if there exists a set of variables $\mathbf{Z}$ such that

$$Pr[X_i \mid X_j, \mathbf{Z}] = Pr[X_i \mid \mathbf{Z}]$$

There is an edge between two variables if and only if there is no set of variables that makes them conditionally independent. We refer to the endpoints of the edge as parent and child, where the edge is directed from the parent to the child.

The structure implies a factorization of the joint probability distribution of the variables

$$Pr[X_1, \ldots, X_n] = \prod_{i=1}^{n} Pr[X_i \mid \pi(X_i)]$$

where $\pi(X_i)$ denotes the set of parents of X_i in G. A given structure defines a family of distributions. To encode a particular probability distribution, we must specify the local probability distributions $Pr[X_i \mid \pi(X_i)]$. Both the structure and the local probability distributions (parameters) can be learned from data, see e.g., [Heckerman et al., 1995] for more details. In this paper, we focus on methods for learning the structure of networks from data.

2.1 Structure Learning Algorithms

The task of structure learning is to recover the Bayesian network structure from a set of samples drawn from the distribution. There are two primary types of structure learning algorithms identified in the literature: search-and-score and constraint-based approaches. The algorithms we consider here are designed for fully observed samples of discrete data. Buntine [1996] provides a review of the literature.

The search-and-score approach poses structure learning as an optimization problem. These techniques use a search algorithm (such as greedy local search) to to find a network that optimizes the scoring function within the vast (super-exponential) space of possible Bayesian networks. The scoring function measures how well the network fits the training data. Possible scoring functions include the *a posteriori* probability of the network, penalized likelihood scores, and information theoretic measures [Buntine, 1996]. In this paper, the algorithms use the popular BDeu metric to score structures during learning [Heckerman et al., 1995]. Under certain conditions, the BDeu score corresponds to the *a posteriori* probability of the network structure given the training data. This score is also likelihood equivalent. It assigns the same probability to networks that encode the same distribution, meaning some edges can be reversed without affecting the score.

Constraint-based approaches use either a statistical test or information-theoretic measure to assess the level of conditional independence between variables [Spirtes et al., 2000]. As each conditional independence constraint is identified, only networks that are consistent with the current set of constraints set are retained.

2.2 Evaluating the Quality of Learned Structures

A commonly used measure of performance is log-likelihood on sample test data. Given a fixed network structure and a set of training instances, the posterior distribution of the joint probability has a closed form solution (under the same assumptions as those underlying the BDeu metric) [Heckerman et al., 1995].

The structural Hamming distance (SHD) is a measure that compares the learned structure to the "gold standard" structure that generated the data [Tsamardinos et al., 2006]. SHD is the total of three types of errors – extra edges, missing edges, and reversed edges. It treats edges uniformly, even though some may connect only weakly dependent variables and have little effect on the probability distribution encoded by the network.

We can treat the distribution encoded by a structure as a function of the training data and decompose its performance on test data into bias and variance. The bias of the structure is analogous to a weighted SHD — edges are weighted by the strength of dependence between the variables it connects — although bias does not penalize for excess structure. This is captured by variance, which measures structure's sensitivity to the training data.

During learning, algorithms are selecting structure that balances bias and variance. Relating this to the operators of the search-and-score algorithms: adding an edge can only reduce bias and increase variance. Deleting an edge or not adding an edge that is in the true structure increases bias and reduces variance. Changing the direction of an edge can affect both bias and variance, depending on the other edges in the structure. Scoring functions like BDeu score attempt to control for variance by penalizing complexity (in terms of number of parameters).

2.3 Max-Min Hill Climbing

The Max-Min Hill Climbing (MMHC) algorithm combines both the search-and-score and constraint-based techniques into a single hybrid algorithm [Tsamardinos et al., 2006]. The first phase of MMHC identifies an undirected graph called a 'skeleton' which has an edge for each pair of dependent variables. To construct the skeleton, the algorithm iterates over possible neighbors and assesses the minimum association between the target node and the candidate neighbor. The Max-Min Heuristic is used to select the variable that maximizes the the minimum association with the target node at each step. Once all possible neighbors have been identified, candidate neighbors that are conditionally independent of the target node are removed.

Conditional independence can be assessed using a statistical test on the training data. This test assumes independence and rejects the null hypothesis when two variables can be shown to be conditionally dependent. The conditional independence tests are not run when the counts of the cells of the corresponding contingency table do not exceed a specified threshold, in this case when the average number of instances per parameter is below five. When the threshold is not met, dependence is assumed between the target node and the candidate neighbor.

The statistical power of these tests depend on the amount of data, level of association, and the p-value cutoff uses to assess dependence. Given two variables that are conditionally dependent, this test will make the right choice under two conditions. If there is sufficient data to show the association, it will reject the null hypothesis at the given p-value. If there is insufficient data (according to the threshold), it will elect not to run the test and (correctly) assume dependence. However, if association is weak and/or the sample size small, the test may fail to reject the null hypothesis (type II error) and assume independence.

The second phase of MMHC is used to select which edges will appear in the final network and to orient the edge directions based on the network skeleton identified in phase 1. MMHC uses a greedy search that is constrained to edges identified during phase 1, no additional edges can be added though not all edges in the skeleton may be chosen by greedy search. The algorithm uses the BDeu score as the scoring metric (with a uniform prior on network structure).

3 Hypotheses

There are at least three possible explanations for why MMHC performs worse, as measured by log-likelihood, than Greedy Search. In this section we outline these hypotheses and make connections, where appropriate, to previous work.

3.1 Low Power

The first hypothesis is that the skeleton-construction phase of MMHC commits errors and thus the constraints imposed on the search space eliminate the structures that could achieve the highest likelihood given the training data.

The skeleton network produced by the first phase of MMHC can contain two kinds of errors. A false positive error is an edge in the skeleton connecting variables that are conditionally independent. Such errors stem from two causes: a type I error in the statistical test (rejecting the null hypothesis of independence when it is true), and by default when the algorithm elects not to apply the test in cases of data sparsity.

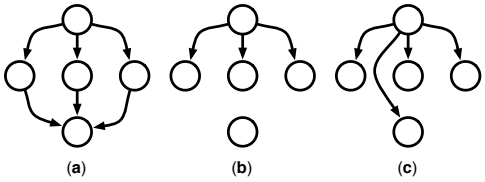

Figure 1: A simple example illustrating the bias-variance tradeoff in structure learning. Assuming the data is sampled from structure (a), this structure has no bias and high variance. Networks (b) and (c) are lower variance, biased approximations. Network (c) might represent the best tradeoff.

False positives only expand the search space (making the space MMHC considers more closely resemble the space of Greedy Search) and therefore cannot explain why MMHC finds lower performing structures than Greedy Search.

False negative error is the absence of an edge between dependent variables. Despite the claims of Tsamardinos et al. [2006], such errors can (and do) occur. They are caused by type II error in the statistical test (discussed in Section 2.3). This behavior occurs when the sample is too small to detect dependence, but large enough to pass the arbitrary threshold of 5 instances per parameter. In this region, the statistical test has low power. The search phase of MMHC cannot correct false negative errors because the search only considers edges that are present in the skeleton. False negatives could be the cause of the lower performance of MMHC, because the search phase cannot correct these errors since it only considers the edges in the skeleton.

3.2 High Variance

Assuming the errors committed in the first phase do not have a substantial impact on performance, a second explanation is that the best structures lie outside the constrained search space. Because the structure that generated the data lies within the constrained space, this hypothesis implies that there are better structures than the generating one.

Whether this occurs depends, in part, on the number of training samples and the complexity of the network and is best understood by considering the bias and variance of the structure and how they impact performance. Consider the example in Figure 1. The generating structure (Fig. 1 (a)) has no bias, so its expected performance is a function solely of its variance. As sample size decreases, the variance (and consequently error) increases. It is possible that there is a better performing structure that trades off a small amount of bias for a large reduction in error. Two examples are shown: Figure 1 (b) certainly has lower variance, but may have too much bias and Fig. 1 (c) may be the best tradeoff. Observe that the network in Figure 1 (c) includes an edge between variables that are conditionally independent (in the true structure), which means that it lies outside the space of networks considered by MMHC, assuming the algorithm makes no Type I errors.

3.3 Hard Search

The final hypothesis is that the constrained search space does contain good structures, but this space is harder to navigate than the unconstrained space. We outline two possible ways in which the constraints could interact with search.

3.3.1 Overfitting

The BDeu metric, which is used to score a candidate structure during search, penalizes for complexity using an arbitrary parameter (the equivalent sample size). If this parameter is set too low, the structure can overfit the data. Since Greedy Search and MMHC use the same scoring metric, this would affect both algorithms. However, the magnitude of the effect could be partially determined by the search space such that it is more severe for MMHC.

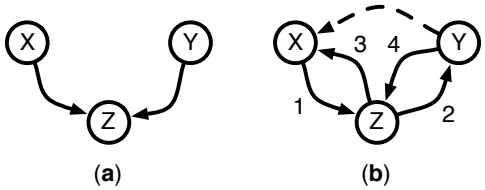

Figure 2: This example illustrates how constrained search may get stuck at a local maxima. The true structure is (a) and we depict the search process in (b). Starting from an empty graph, the BDeu score of adding edge (1) is the same as the score of adding (3); similarly for edges (2) and (4). If the algorithm chooses randomly, one quarter of the time edges (2) and (3) will be selected. In the constrained space, this is a local maxima; in the unconstrained space, the dashed edge can be added to capture the dependency between X and Y given Z.

3.3.2 Misdirected Edges

A virtue of the BDeu metric is that if two structures are equivalent (assert the same set of conditional independence relations) then they have equivalent BDeu scores. We hypothesize that in certain situations, this virtue can become a vice. Because the score is agnostic about edge direction, a sequence of search operations can lead to a structure that is sub-optimal, but cannot be improved with a single operation. When search evaluates adding an edge between two endpoints, the edge will receive the same score in either direction (unless it is compelled by other edges incident to the endpoints). The search algorithm chooses randomly among successors having tied scores. If it breaks ties incorrectly, then it can end up at a state where no single edge modification can result in a better score. An example is shown in Figure 2. Observe that these states may not be local maxima in the unconstrained search space, because "repair" edges can be added between the parents of a child.

4 Experimental Design

To replicate the findings of Tsamardinos et al. [2006], we base several facets of our experimental design (datasets, sample sizes, and some performance measures) on their approach. Below we describe the networks and the datasets sampled from the networks. Then we discuss evaluation metrics. Finally we discuss the algorithms, all of which are variations on greedy search. The networks, data, and algorithms are publicly available at [Hay et al., 2007].

4.1 Networks and Datasets

We evaluate the structure learning algorithms on five Bayesian networks that cover a number of different domains such as medical diagnosis, insurance risk, meteorology, and agriculture (see Table 1). This is a subset of the networks evaluated in Tsamardinos et al. [2006]. We limited our study to the networks that (a) were available in the Bayesian network repository [Bayesian Network Repository, 1997] and (b) were relatively small (under 100 variables) due to the computational expense of Greedy Search. For each network, we generated random samples of varying size (500, 1000, 2000, and 5000) for use in training. To assess how well the learned structure approximates the generating distribution, we measured the log-likelihood of a large test sample (500,000 instances). Performance measurements were averaged over five training samples at each sample size. This leads to a total of 100 trials (5 datasets, 4 sample sizes, 5 runs for each dataset-sample size combination).

4.2 Algorithms

The two main algorithms under investigation are unconstrained greedy search (uGS) and MMHC, which we denote in this context as $cGS(\widehat{neighbors})$. This notation indicates that the algorithm performs a greedy search constrained to add edges only between neighbors, which in the case of MMHC are estimated from data.

Table 1: A Summary Of The Bayesian Networks Used.

Network	No. Vars.	No. Edges	In/Out Degree	Domain Size (Avg.)
Alarm	37	46	4/5	2-4 (2.8)
Barley	48	84	4/5	2-67 (8.8)
Hailfinder	56	66	4/16	2-11 (4.0)
Insurance	27	52	3/7	2-5 (3.3)
Mildew	35	46	3/3	3-100 (17.6)

To differentiate between the hypotheses, we also analyze the performance of several variants of constrained greedy search. The variations come from allowing the search to consult an oracle, who knows the structure of the Bayesian network from which the data is sampled. The algorithm $cGS(neighbors)$ is identical to $cGS(\widehat{neighbors})$ except that the correct neighbors are provided by the oracle. $cGS(parents)$ can construct any network that contains a subset of the edges in the generating network. It is identical to $cGS(neighbors)$ except that edge direction is also specified by the oracle. Analogously, $cGS(ancestors)$ is the same as uGS except edge direction is specified by the oracle. Finally, $cGS^*(neighbors)$ can in principle add an edge in either direction between any pair of neighboring vertices. However, the algorithm consults the oracle when it needs to break ties among successor states: if the best operation is to connect two variables, but the score is the same whether the edge is oriented from parent to child or child to parent, the algorithm consults the oracle. If there is no tie, i.e., one direction scores higher than the other, the edge will be added even if it is misdirected.

5 Experimental Results

For exposition purposes, we present results specific to each hypothesis. A complete assessment of performance is available in the Appendix.

5.1 Low Power

To evaluate the low-power hypothesis, we compare the performance of $cGS(neighbors)$ to the performance of $cGS(\widehat{neighbors})$ and uGS. Because $cGS(neighbors)$ consults an oracle to build the skeleton network, the skeleton contains no missing edges and so has no errors due to low power. Figure 3 compares the log-likelihood of the three algorithms. As the figure illustrates, on some datasets (Alarm, Barley, Insurance), having the correct skeleton provides substantial reductions in error. However, on all datasets uGS outperforms $cGS(neighbors)$, indicating that constraining the search space leads to lower performance. The results on the Mildew data are anomalous in that $cGS(neighbors)$ performs significantly worse than $cGS(\widehat{neighbors})$, meaning that correcting the errors in the neighbor sets actually hurt performance. Because of the high cardinality of many the variables in Mildew, there is not enough data for $cGS(\widehat{neighbors})$ to perform conditional independence tests. Thus, on this dataset, $cGS(\widehat{neighbors})$ adds few constraints to the search space.

5.2 High Variance

To illustrate the effect of variance, Figure 4 compares the log-likelihood performance of the structure that generated the data to the structures learned by the algorithms. We emphasize that this is distinct from the likelihood of the generating distribution, which has the highest likelihood on the test data (indicated by `true BN` in the figures). In this context, the learned structure is treated as a function of the training data; the distribution encoded depends on what samples are observed in the training data. At small sample sizes in all of the datasets except Alarm, the true structure has lower performance than the learned structures. Since the true structure is unbiased, the low performance is due entirely to variance.

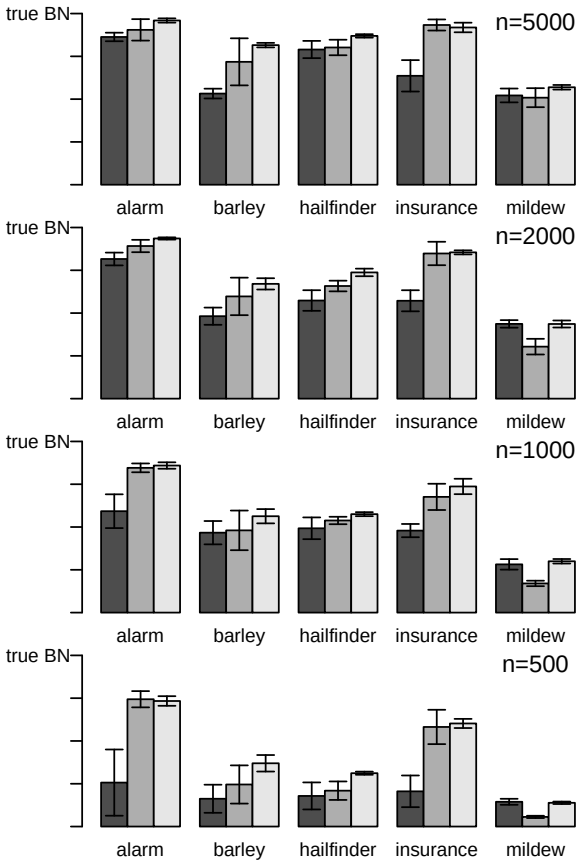

Figure 3: A comparison of three algorithms — from left to right $cGS(\widehat{neighbors})$, $cGS(neighbors)$, uGS — in terms log-likelihood performance on five datasets at four different sample sizes. Each bar represents an average (with standard error also shown) over five training sets. Each subplot is scaled based on the likelihood of the data given the true Bayesian network. The substantial gain in performance between $cGS(\widehat{neighbors})$ and $cGS(neighbors)$ provides evidence for the hypothesis that $cGS(\widehat{neighbors})$ makes significant errors due to low power.

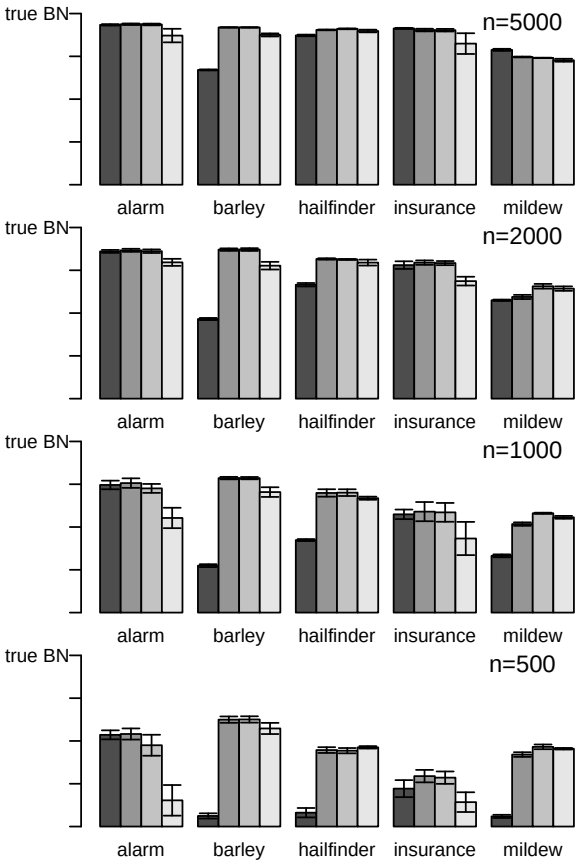

Figure 4: A comparison of the log-likelihood of the true structure, $cGS(parents)$, $cGS(ancestors)$, uGS. The low performance of the true structure at small samples indicates high variance. On most datasets, high-performing structures are found both within the constrained space ($cGS(parents)$) and outside ($cGS(ancestors)$).

At small samples, variance becomes a factor and the second hypothesis is concerned with whether structures with low variance and high performance lie within the constrained search space. We analyze this question by comparing two algorithms that are well equipped to trade off reduction in variance with increase in bias. They differ in that one ($cGS(parents)$) is confined to the search space that MMHC is designed to search and the other ($cGS(ancestors)$) is unrestricted, much like Greedy Search. Both consult an oracle to direct the edges, thereby factoring out any error due to edge misdirection. Results are shown in Figure 4. On four of the five datasets, $cGS(parents)$ and $cGS(ancestors)$ are the highest performing algorithms and their performances are indistinguishable from one another. Both contain substantially fewer edges than the true structure, yet this increase in bias is clearly worth the reduction in variance that it provides. On these datasets, the results indicate that good structures lie within the constrained search space that MMHC is designed to search.

On the fifth dataset, Mildew, we observe that $cGS(parents)$ performs significantly worse than both

$cGS(ancestors)$ and uGS. This suggests that in this domain, achieving a good balance between bias and variance involves more than simply dropping edges from the true structure. $cGS(ancestors)$ is able to reduce the bias, without greatly increasing variance, by adding in a few key extra edges. These findings are consistent with the fact that Mildew appears to be the most complex probability distribution, given that its variables have high domain cardinalities and fan-in (no. of parents) comparable with the other networks.

5.3 Hard Search

The first two hypotheses cannot fully explain the gap in performance between $cGS(neighbors)$ and uGS. We next evaluate the remaining hypothesis that the constrained space is harder to search than the unconstrained space.

5.3.1 Overfitting

To assess the overfitting hypothesis, we compare $cGS(neighbors)$ and uGS both in terms of their performance on the test data and their BDeu scores on the training data. If the network that $cGS(neighbors)$ produces has a higher score on the training data than the network produced by uGS, yet has lower performance on the test data, then overfitting has occurred. This event occurred only in 2 out of 100 trials, strongly suggesting that overfitting is not the cause.

5.3.2 Misdirected Edges

Our final hypothesis — that the search misdirects edges and cannot recover from these errors in the constrained space — is evaluated by empowering $cGS(neighbors)$ with an ability to distinguish edge orientation under certain conditions. The $cGS^*(neighbors)$ algorithm consults the oracle whenever it needs to break ties among successor states: if the best operation is to connect two variables, but the score is the same whether the edge is oriented from parent to child or child to parent, the algorithm consults the oracle. (This is distinct from $cGS(parents)$, which does not even consider misdirected edges.) The $cGS^*(neighbors)$ algorithm does well, with performance comparable to or better than uGS, as shown in Figure 5.

5.4 Discussion of Results

To summarize, the results indicate that MMHC's independence tests are far from perfect, but that correcting those tests alone cannot account for the performance gap. Furthermore, they show conclusively that high-performing structures for small samples differ significantly from the true structure. Though those structures exist within the space searched by MMHC, the algorithm does not routinely find them. Correcting errors in edge detection closes the performance gap by successfully keeping the search algorithm away from local maxima.

6 Improved Algorithm

As the experimental results indicate, the MMHC algorithm can be improved along two dimensions. The first is adjusting for low statistical power. We focus here on the second area of improvement: modifying the search phase so that it has an opportunity to "repair" its mistakes in edge misdirection. The constraints learned in the first phase of the algorithm combined with the (unavoidable) mistakes in edge direction have the effect of boxing in search such that with only the search operations of adding, deleting or removing a single edge, search cannot escape local maxima.

Several remedies are possible for this second problem. Expanding the set of search operators by including operations that simultaneously modify multiple edges may help. More complex operators are considered by Chickering [2002] and Moore and Wong [2003].

Second, the constraints learned in the first phase of the algorithm could be revised during search so that they reflect the conditional independence assertions of the current network structure. A third remedy is to relax the constraints altogether once the algorithm reaches a local maximum, performing an unconstrained search from where the constrained search leaves off.

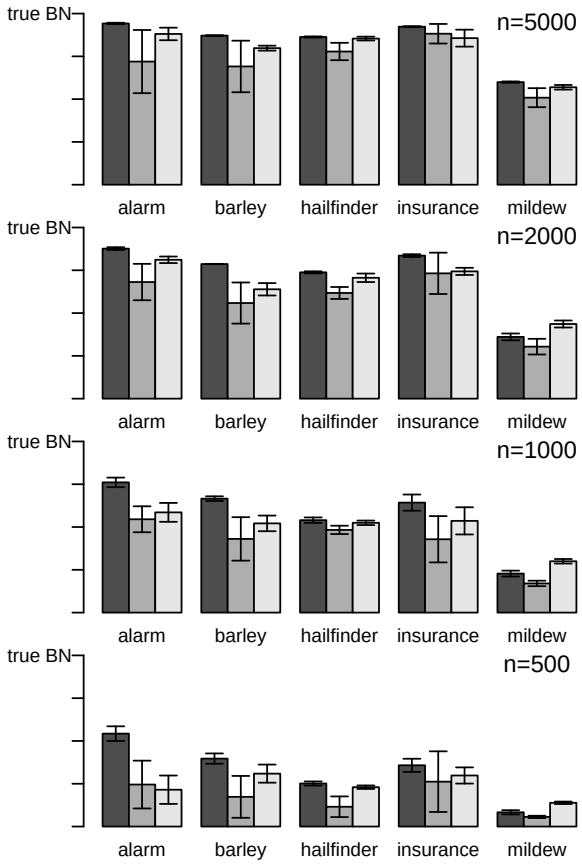

Figure 5: This plot shows likelihood for the algorithms, from left to right, $cGS^*(neighbors)$, $cGS(neighbors)$, uGS. The results illustrate how the remaining performance gap between $cGS(neighbors)$ and uGS can be closed by resolving ambiguity about edge orientation.

We evaluated this last remedy based on two goals: we desire a solution should have the accuracy of Greedy Search (measured here by log-likelihood), yet the efficiency of MMHC (measured here by runtime). Table 2 depicts the results. The remedy is denoted as $MMHC^+$. The table presents the runtime of both Greedy Search and $MMHC^+$ *normalized by the runtime of MMHC* and the likelihood of $MMHC^+$ and MMHC *normalized by the likelihood of Greedy Search*. Thus a number less than or equal to 1 indicates that $MMHC^+$ is matching the performance of the better algorithm *for that measure*. The results are encouraging, in that the remedy has the accuracy of Greedy Search yet is considerably more efficient. However, it remains slower than MMHC, in some cases by an order of magnitude. Furthermore, these networks are fairly small (less than one hundred variables) and the solution may not scale to larger networks as well as MMHC has been shown to do.

7 Conclusions

One conclusion we draw from this analysis is a commentary on the objective of structure learning. As indicated by the use of measures such as SHD, many view the objective as learning the structure of the distribution that generated the data. We argue that the objective should be to learn the best structure *given the size of the sample*. The best structure is not always the generating structure, because its posterior distribution may have high variance. Thus evaluating algorithms based on the extent to which their output matches the generating network can cloud the picture. We emphasize that this is not a critique of constraint-based algorithms, but an acknowledgment that for the algorithm to find high-performance

Table 2: Timing and likelihood results evaluating the proposed remedy to the search challenges faced by MMHC. The measures are normalized such that a score of one indicates performance on par with the best algorithm for that measure.

Network	Time		
	$MMHC^+$	GS	MMHC
Alarm	10.00	15.58	1.00
Barley	3.52	8.32	1.00
Hailfinder	2.94	6.33	1.00
Insurance	3.69	5.33	1.00
Mildew	1.11	1.34	1.00
	Inverse Likelihood		
	$MMHC^+$	GS	MMHC
Alarm	0.93	1.00	3.69
Barley	1.15	1.00	1.9
Hailfinder	1.00	1.00	1.26
Insurance	1.05	1.00	2.65
Mildew	0.99	1.00	1.02

structures, it must gracefully approximate its objective at small sample sizes. As we observe in the MMHC algorithm, the constraints can interact with search in some unexpected ways, leading to lower performance than unconstrained search. There are remedies as we have shown and in future work, we intend to explore them in more depth. Finally, our conclusions suggest that any knowledge which constrains the search space of Bayesian net structures, even if it is accurate, could reduce the likelihood of the learned network.

Acknowledgments

The authors wish to thank Laura Brown for clarifying some aspects of the MMHC implementation, and Cindy Loiselle for helpful comments. This research is supported by NSF under contract number IIS-0326249. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of NSF or the U.S. Government.

References

Bayesian Network Repository, 1997. URL `http://www.cs.huji.ac.il/labs/compbio/Repository/`.

W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. on Knowledge and Data Eng.*, 8(2):195–210, 1996.

D. Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

M. G. Hay, A. Fast, and D. Jensen. Supplemental materials. URL `http://www.cs.umass.edu/~mhay/uai2007/`. 2007.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3), 1995.

A. Moore and W.-K. Wong. Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *Proc. of the 20th Intl. Conf. on Machine Learning*, 2003.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann, 1998.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. MIT Press, 2000.

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.*, 65(1):31–78, 2006.

A Complete Results

Tables 3-7 summarize the log-likelihood performance of all the algorithms. Each value shown is the average log-likelihood over five different training datasets.

Table 3: Comparison of Log-likelihood on Alarm.

Algorithm	**Log-likelihood**			
SampleSize	500	1000	2000	5000
generating distribution	-5249834.689	-5249834.689	-5249834.689	-5249834.689
generating structure	-5394927.109	-5329284.13	-5293447.054	-5270458.703
$cGS(parents)$	-5393395.165	-5325888.933	-5291459.512	-5269345.863
$cGS(ancestors)$	-5413906.855	-5335577.956	-5293112.573	-5269749.651
$cGS(neighbors)$	-5504311.931	-5403162.07	-5357283.585	-5344450.835
$cGS^*(neighbors)$	-5403990.054	-5330496.137	-5291459.512	-5269345.863
$cGS(\widehat{neighbors})$	-5988050.293	-5654636.86	-5432599.99	-5385686.447
$MMHC^+$	-5525506.293	-5393923.617	-5313667.113	-5278489.472
uGS	-5514515.974	-5389569.768	-5313457.486	-5290023.568

Table 4: Comparison of Log-likelihood on Barley.

Algorithm	**Log-likelihood**			
SampleSize	500	1000	2000	5000
generating distribution	-24161905.91	-24161905.91	-24161905.91	-24161905.91
generating structure	-32750557.33	-30806363.71	-29057663.7	-27176643.62
$cGS(parents)$	-27598369.11	-26122188.2	-25344161.39	-24903437.13
$cGS(ancestors)$	-27589521.53	-26124796.82	-25341051.95	-24904925.51
$cGS(neighbors)$	-28839255.6	-27384133.54	-26658930.52	-25910972.57
$cGS^*(neighbors)$	-27574075.63	-26052713.1	-25369517.63	-24890539.27
$cGS(\widehat{neighbors})$	-29356008.95	-27467644.6	-27375134.07	-27060733.81
$MMHC^+$	-28081502.93	-26853067.25	-26160473.79	-25429556.24
uGS	-28071062.5	-26871179.49	-26204322.65	-25305512.44

Table 5: Comparison of Log-likelihood on Hailfinder.

Algorithm	**Log-likelihood**			
SampleSize	500	1000	2000	5000
generating distribution	-24556047.51	-24556047.51	-24556047.51	-24556047.51
generating structure	-25941735.34	-25424878.9	-25060170.93	-24748628.49
$cGS(parents)$	-25390151.32	-25009928.96	-24833091.09	-24699908.27
$cGS(ancestors)$	-25395489.4	-25007043.78	-24837502.67	-24690584.63
$cGS(neighbors)$	-25486332.43	-25099589.74	-24958362.12	-24789338.1
$cGS^*(neighbors)$	-25343405.41	-25039862.14	-24831744.79	-24699908.27
$cGS(\widehat{neighbors})$	-25522514.51	-25153273.92	-25058144.46	-24803284.29
$MMHC^+$	-25382718.23	-25065256.88	-24868981.4	-24685110.52
uGS	-25366100.53	-25056041	-24865055.39	-24709709.08

Table 6: Comparison of Log-likelihood on Insurance.

Algorithm	**Log-likelihood**			
SampleSize	500	1000	2000	5000
generating distribution	-6543159.764	-6543159.764	-6543159.764	-6543159.764
generating structure	-6815442.149	-6692112.909	-6620018.142	-6573588.417
$cGS(parents)$	-6789882.433	-6686592.63	-6614929.167	-6576866.762
$cGS(ancestors)$	-6792998.65	-6688048.39	-6615901.498	-6577141.369
$cGS(neighbors)$	-6858464.296	-6787422.369	-6657643.984	-6593527.387
$cGS^*(neighbors)$	-6817613.921	-6695915.881	-6613391.331	-6575902.793
$cGS(\widehat{neighbors})$	-7142084.918	-6936003.099	-6865994.987	-6817494.271
$MMHC^+$	-6876608.101	-6723817.319	-6640856.95	-6600424.895
uGS	-6843069.284	-6741495.557	-6652735.146	-6604564.169

Table 7: Comparison of Log-likelihood on Mildew.

Algorithm	**Log-likelihood**			
SampleSize	500	1000	2000	5000
generating distribution	-19539886.84	-19539886.84	-19539886.84	-19539886.84
generating structure	-28432609.82	-25858455.11	-23559877.02	-21544543.28
$cGS(parents)$	-25018013.99	-24104541.24	-23376189.09	-21938155.52
$cGS(ancestors)$	-24592510.3	-23508470.37	-22785801.66	-21989533.88
$cGS(neighbors)$	-25190133.96	-24506863.16	-23707861.82	-22482429.37
$cGS^*(neighbors)$	-25028839.46	-24161792.15	-23367465.54	-21938279.1
$cGS(\widehat{neighbors})$	-24661412.66	-23839751.54	-22912020.33	-22404843.31
$MMHC^+$	-24661412.66	-23727717.46	-22821545.3	-22150557.25
uGS	-24696935.94	-23731812.86	-22915396.78	-22118569.19