Bayesian Network Induction via Local Neighborhoods

Dimitris Margaritis Sebastian Thrun

August 1999
CMU–CS–99–134

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In recent years, Bayesian networks have become highly successful tool for diagnosis, analysis, and decision making in real-world domains. We present an efficient algorithm for learning Bayesian networks from data. Our approach constructs Bayesian networks by first identifying each node's Markov blankets, then connecting nodes in a consistent way. In contrast to the majority of work, which typically uses hill-climbing approaches that may produce dense nets and incorrect structure, our approach typically yields consistent structure and compact networks by heeding independencies in the data. Compact networks facilitate fast inference and are also easier to understand. We prove that under mild assumptions, our approach requires time polynomial in the size of the data and the number of nodes. A Monte Carlo variant, also presented here, is more robust and yields comparable results at much higher speeds.

Keywords: Bayesian networks, Bayesian network induction, independence tests, Markov blanket, causal discovery.

1 Introduction

A great number of scientific fields today benefit from being able to automatically estimate the probability of certain quantities of interest that may be difficult or expensive to observe directly. For example, a doctor may be interested in estimating the probability of heart disease from indications of high blood pressure and other directly measurable quantities. A computer vision system may benefit from a probability distribution of buildings based on indicators of horizontal and vertical straight lines. Probability densities proliferate the sciences today and advances in its estimation are likely to have a wide impact on many different fields.

Bayesian networks are a succinct and efficient way to represent a joint probability distribution among a set of variables. As such, they have been applied to fields such as those mentioned [HC91, Ago90]. Besides their ability for density estimation, their semantics lend them to what is sometimes loosely referred to as *causal discovery*, namely directional relationships among quantities involved. It has been widely accepted that the most parsimonious representation for a Bayesian net is one that closely represents the causal independence relationships that may exist. For these reasons, there has been great interest in automatically inducing the structure of Bayesian nets automatically from data, preferably also preserving the independence relationships in the process.

Two research approaches have emerged. The first employs independence properties of the underlying network that produced the data in order to discover parts of its structure. This approach is mainly exemplified by the SGS and PC algorithms [SGS93] as well as for restricted classes such as trees [CL68] and polytrees [RP89]. The second approach is concerned more with data prediction, disregarding independencies in the data. It is typically identified with a hill-climbing or best-first beam search in the space of possible structures, employing data likelihood as a scoring function. The result is a local maximum likelihood network structure for representing the data, and is one of the more popular techniques used today.

This paper presents an approach that belongs in the first category. It addresses the two main shortcomings of the prior work which, we believe, are preventing its use from becoming more widespread. These two disadvantages are: exponential execution times, and proneness to errors in dependence tests used. The former problem is addressed in this paper in two ways. One is by identifying the local neighborhood of each variable in the Bayesian net as a preprocessing step, in order to facilitate the recovery of the local structure around each variable in polynomial time under the assumption of bounded neighborhood size. The Monte Carlo version of the algorithm goes one step further, employing a constant number of randomized tests in order to ascertain the same result with high probability. The second disadvantage of this research approach, namely proneness to errors, is also addressed by the randomized version, by using multiple data sets (if available) and Bayesian accumulation of evidence.

2 Preliminaries

In the following, regular variables are in capitals while capital bold-faced letters indicate sets. $\mathbf{V}$ is the set of variables under consideration, which coincides with the set of nodes in the corresponding Bayesian net. The set of edges is denoted by $\mathbf{E}$. Letters n and m denote the set sizes $|\mathbf{V}|$ and $|\mathbf{E}|$, respectively. All variables are assumed to be visible (*i.e.* observable). The notation $X \leftrightarrow_{\mathbf{S}} Y$ denotes that variables X and Y are dependent upon conditioning on the variables in the set $\mathbf{S}$, while $X \not\leftrightarrow_{\mathbf{S}} Y$ indicates conditional independence.

In the following, we also assume that the reader is familiar with the concept of d-separation in the Bayesian net framework. For a description of this, see [Pea88].

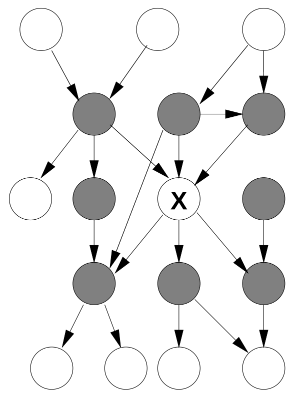


Figure 1: Example of a Markov blanket of variable X. The members of the blanket are shown shaded.

1. $\mathbf{S} \leftarrow \emptyset$.
2. While $\exists Y \in \mathbf{V} - \{X\}$ such that $Y \leftrightarrow_{\mathbf{S}} X$, do $\mathbf{S} \leftarrow \mathbf{S} \cup \{Y\}$. **[Growing phase]**
3. While $\exists Y \in \mathbf{S}$ such that $Y \not\leftrightarrow_{\mathbf{S}-\{Y\}} X$, do $\mathbf{S} \leftarrow \mathbf{S} - \{Y\}$. **[Shrinking phase]**
4. $\mathbf{B}(X) \leftarrow \mathbf{S}$.

Table 1: The Markov Blanket Algorithm.

3 The Grow-Shrink Markov Blanket Algorithm

The concept of the *Markov blanket* of a variable or a set of variables is central to this paper. The idea itself is not new (for example, see [Pea88]). It is surprising, however, how little attention it has attracted in the context of Bayesian net structure learning for all its being a fundamental property of a Bayesian net. The definition of a Markov blanket is as follows: for any variable $X \in \mathbf{V}$, the Markov blanket $\mathbf{BL}(X) \subseteq \mathbf{V}$ is any set of variables such that for any $Y \in \mathbf{V} - \mathbf{BL}(X) - \{X\}$, $X \not\leftrightarrow_{\mathbf{BL}(X)} Y$. In other words, $\mathbf{BL}(X)$ completely shields variable X from any other variable outside $\mathbf{BL}(X) \cup \{X\}$. The notion of a *minimal Markov blanket*, called a *Markov boundary*, is also introduced in [Pea88] and its uniqueness shown under certain conditions. The Markov boundary is not unique in certain pathological situations, such as the equality of two variables. In our following discussion we will assume that the conditions necessary for its existence and uniqueness are satisfied and we will identify the Markov blanket with the Markov boundary, using the notation $\mathbf{B}(X)$ for the blanket of variable X from now on. It is also illuminating to mention that, in the Bayesian net framework, the Markov blanket of a node X is easily identifiable from the graph: it consists of all parents, children and parents of children of X. An example Markov blanket is shown in Fig. 1. Note that any of the blanket nodes, say Y, is dependent with X given $\mathbf{B}(X) - \{Y\}$.

In Table 1 we present an algorithm for the recovery of the Markov blanket of X based on pairwise independence tests. It consists of two phases, a growing and a shrinking one. The growing phase adds variables to a growing set $\mathbf{S}$ as long as they are dependent with X given the current contents of $\mathbf{S}$. The idea behind this is simple: as long as the Markov blanket property of X is violated (*i.e.* there exists a variable in $\mathbf{V}$

that is dependent on X), we add it to the current set $\mathbf{S}$ until there are no more such variables. In this process however, there may be some variables that were added to $\mathbf{S}$ that were really outside the blanket. Such variables are those that have been rendered independent from X at a later point when "intermediate" nodes of the underlying Bayesian net were added to $\mathbf{S}$. This observation motivates the shrinking phase, which identifies and removes these variables.

Below we prove the correctness of the algorithm and present a timing analysis. By "correctness" we mean the property of the algorithm to produce the correct (original) Bayesian net if all independence tests done during its course are assumed to be correct. A related issue is *stability*, which investigates the effect (small or large) of errors in those tests to the resulting net structure.

Proof of correctness

We assume that all dependence tests are correct in the following proof.

There does not exist any variable $Y \in \mathbf{B}(X)$ at the end of the growing phase that is not in $\mathbf{S}$. The proof is by contradiction: take $Y \in \mathbf{V}$ such that $Y \in \mathbf{B}(X)$ but $Y \notin \mathbf{S}$. Then either Y is a direct neighbor of X (direct ancestor or direct descendant), or it is a direct ancestor of a direct descendant of X. In the former case $Y \leftrightarrow_{\mathbf{T}} X$ for any $\mathbf{T} \subseteq \mathbf{V}$. Therefore necessarily $Y \in \mathbf{S}$ at the end of the growing phase. In the latter case, either the common child of X and Y, say Z, is in $\mathbf{S}$ or is not. If $Z \in \mathbf{S}$ then Y must also be in $\mathbf{S}$, since $X \leftrightarrow_{\mathbf{S}} Y$. If $Z \notin \mathbf{S}$ then we have a contradiction by the same argument as above since Z is a direct descendant of X.

For the shrinking phase, we have to prove two things: that we never remove any variable Y from $\mathbf{S}$ if $Y \in \mathbf{B}(X)$, and that all variables $W \notin \mathbf{B}(X)$ are removed.

For the former, suppose Y is the first variable in $\mathbf{B}(X)$ that we are attempting to remove. Then Y is either a direct neighbor of X or the direct ancestor of a direct descendant of X, say Z. In the former case, since $Y \leftrightarrow_{\mathbf{T}} X$ for any $\mathbf{T} \subseteq \mathbf{V}$, Y cannot be removed, leading to a contradiction. In the latter case, since Y is the first variable in $\mathbf{B}(X)$ to be removed, then Z must be in $\mathbf{S}$ and therefore, since $Y \leftrightarrow_{\mathbf{S}} X$, Y cannot be removed.

Finally we need to show that there is no variable W in $\mathbf{S}$ at the end of the thinning phase such that $W \notin \mathbf{B}(X)$. Suppose the opposite. Then since $\mathbf{B}(X) \subseteq \mathbf{S}$ as shown above, then $W \not\leftrightarrow_{\mathbf{S}} X$, and W will necessarily be removed by the algorithm during this phase.

Timing Analysis

Each dependence test takes $O(n\,|\mathbf{D}|)$ time, where $\mathbf{D}$ is the set of examples input to the algorithm. We make the assumption that combinations of variable values that do not appear in the data are improbable and their probability is approximated by 0 in the absence of other information. This assumption makes the test linear in the number of examples (and not exponential in the number of variables as would be the case if examples existed for all possible value combinations of $\mathbf{V}$). Each dependence test uses $O(|\mathbf{D}|)$ space at worst to store the counters for each variable value combination of the conditioning set that appears in the data.

Frequently the number of dependence tests is reported as a measure of the performance of Bayesian net reconstruction algorithms, *e.g.* [SGS93, CBL97]. To determine the number of tests in this algorithm, we assume that the loops in steps 2 and 3 go through eligible variables in an unspecified but fixed order. In step 2, one complete pass through all variables will add at least the parents and children of X. A second pass will add all parents of all children of X, thus including all members of $\mathbf{B}(X)$. A possible third pass will not add any other members and step 2 will terminate. Thus, step 2 conducts $O(n)$ tests at worst. In step 3, a single pass through the variables in $\mathbf{S}$ will remove all members not in $\mathbf{B}(X)$ since $\mathbf{S}$ already contains $\mathbf{B}(X)$.

Therefore the entire algorithm is $O(n)$ in the number of dependence tests.

Discussion

Although the algorithm is already efficient, it may benefit from certain optimizations. One may attempt to use a heuristic in order to add all members of $\mathbf{B}(X)$ in the first pass of step 2, thus eliminating one of the two remaining passes. Using mutual information in the choice of the next variable to examine in step 2, for example, may help improve the performance of the algorithm. The idea is to try to add to $\mathbf{S}$ first these variables for which the mutual information with X is highest. Unfortunately no guarantees can be made of completely eliminating a pass this way because of certain cases where the highest mutual information variables are not members of the blanket. These cases are admittedly not very common.

An opportunity for computational amortization follows from the (easily proven) fact that $Y \in \mathbf{B}(X) \Leftrightarrow X \in \mathbf{B}(Y)$. Using this property, if we are interested in more than one variable's blanket and we obtain them in a sequential manner (as in the GS algorithm described below), we can initialize a variable's blanket, say X, with those variables Y for which we have computed the blanket in previous steps, by checking that they already include X in their blankets.

4 Grow-Shrink (GS) Algorithm for Bayesian Net Induction

The recovery of the local structure around each node is greatly facilitated by the knowledge of the nodes' Markov blankets. What would normally be a daunting task of employing dependence tests conditioned on an exponential number of subsets of large sets of variables—even though most of their members may be irrelevant—can now be focused on the Markov blankets of the nodes involved, making structure discovery much faster and more reliable. We present below the plain version of the GS algorithm that utilizes blanket information for inducing the structure of a Bayesian net. At a later point of this paper, we will present a robust, randomized version that has the potential of being faster and more reliable, as well as being able to operate in an "anytime" manner.

In the following $\mathbf{N}(X)$ represents the direct neighbors of X.

1. [Compute Markov Blankets]
For all $X \in \mathbf{V}$, compute the Markov blanket $\mathbf{B}(X)$.

2. [Compute Graph Structure]
For all $X \in \mathbf{V}$ and $Y \in \mathbf{B}(X)$, determine Y to be a direct neighbor of X if X and Y are dependent given $\mathbf{S}$ for all $\mathbf{S} \subseteq \mathbf{T}$, where $\mathbf{T}$ is the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$.

3. [Orient Edges]
For all $X \in \mathbf{V}$ and $Y \in \mathbf{N}(X)$, orient $Y \to X$ if there exists a variable $Z \in \mathbf{N}(X) - \mathbf{N}(Y) - \{Y\}$ such that Y and Z are dependent given $\mathbf{S} \cup \{X\}$ for all $\mathbf{S} \subseteq \mathbf{U}$, where $\mathbf{U}$ is the smaller of $\mathbf{B}(Y) - \{X, Z\}$ and $\mathbf{B}(Z) - \{X, Y\}$.

4. [Remove Cycles]
Do the following while there exist cycles in the graph:

- Compute the set of edges $\mathbf{C} = \{X \rightarrow Y$ such that $X \rightarrow Y$ is part of a cycle$\}$.
- Remove from the current graph the edge in $\mathbf{C}$ that is part of the greatest number of cycles, and put it in $\mathbf{R}$.

5. [Reverse Edges]
Insert each edge from $\mathbf{R}$ in the graph, reversed.

6. [Propagate Directions]
For all $X \in \mathbf{V}$ and $Y \in \mathbf{N}(X)$ such that neither $Y \rightarrow X$ nor $X \rightarrow Y$, execute the following rule until it no longer applies: If there exists a directed path from X to Y, orient $X \rightarrow Y$.

In the algorithm description above, step 2 determines which of the members of the blanket of each node are actually direct neighbors (parents and children). The way it does that is by a series of dependence tests between X and Y conditioned on all subsets of the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$. Assuming, without loss of generality, that $\mathbf{B}(X) - \{Y\}$ is the smaller set, if any of these tests are successful in separating (rendering independent) X from Y, the algorithm determines that there is no direct connection between them. That would happen when the conditioning set $\mathbf{S}$ includes all parents of X and no common children of X and Y. It is interesting to note the two motivations behind selecting the smaller set to condition on: the first, of course, is speed, but the second stems from reliability; a conditioning set $\mathbf{S}$ causes the data set to be split into $2^{|\mathbf{S}|}$ partitions, so smaller conditioning sets cause the data set to be split into larger partitions and make dependence tests more reliable.

Step 3 of the algorithm exploits the fact that two variables that have a common descendant become dependent when conditioning on a set that includes any such descendant. Since the direct neighbors of X and Y are known from step 2, we can determine whether a direct neighbor Y is a parent of X if there exists another node Z (which would also be a parent) such that any attempt to separate Y and Z by conditioning on a subset of the blanket of Y that includes X, fails (assuming here that $\mathbf{B}(Y)$ is smaller than $\mathbf{B}(Z)$). If the directionality is indeed $Y \rightarrow X \leftarrow Z$, there should be no such subset since, by conditioning on X, there exists a permanent dependency path between Y and Z. This would not be the case if Y were a child of X.

It is possible that an edge direction will be wrongly determined during step 3 due to non-representative or noisy data. This may lead to directed cycles in the resulting graph, which are illegal. It is therefore necessary to remove those cycles by identifying the minimum set of edges than need to be reversed for all cycles to disappear. This problem is closely related to the *Minimum Feedback Arc Set* problem (*MFAS*), which is concerned with identifying a minimum set of edges that need to be removed from a graph that possibly contains directed cycles, in order for all such cycles to disappear. Here instead of removing those edges we want to reverse them. Unfortunately, the *MFAS* problem is NP-complete in its generality [J8̈5] and the weighted edges instance as it applies here is also NP-complete (see theorem in Appendix B). In the algorithm above we introduce a heuristic for its solution that is based on the number of cycles that an edge that is part of a cycle is involved in.

Not all edge directions can be determined during the last two steps. For example, nodes with a single parent or multi-parent nodes (called *colliders*) whose parents are directly connected do not apply to step 3, and steps 4 and 5 are only concerned with already directed edges. Step 6 attempts to ameliorate that. This is done through orienting edges in a way that does not introduce a cycle, if the reverse direction necessarily does. It is not obvious that, for example, if the direction $X \rightarrow Y$ produces a cycle in an otherwise acyclic graph, the opposite direction $Y \rightarrow X$ will not also be involved in some other cycle. However, this is the case. The proof of this is simple is presented below in the proof of correctness.

Proof of correctness

The proof is straightforward. To show that the Bayesian net resulting from the above procedure is equivalent to the original one by which the data was created, we need to show two things:

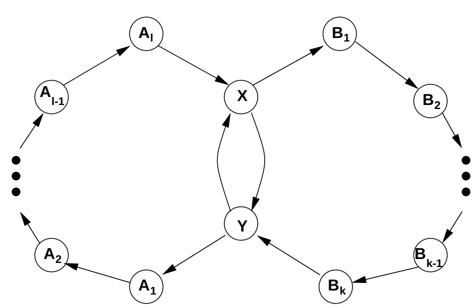
1. All nodes in the resulting net have the same neighbors as the original Bayesian net, and
2. All colliders in the resulting net have the same parents as the original net.

These requirements stem from a theorem (see [VP90]) that states that two Bayesian nets are equivalent if and only if each node has the same neighbors and the two nets have the same "V-structures," i.e. colliders with the same parents (and, by requirement 1, the same children as well).

In the following proof of correctness of step 2, we assume that $|\mathbf{B}(X)| \leq |\mathbf{B}(Y)|$, without loss of generality. Thus tests will involve conditioning on all subsets of $\mathbf{B}(X) - \{Y\}$. To show that the net resulting from the above algorithm satisfies the first requirement, we observe that any two nodes X and Y in the original net that are directly connected remain dependent upon conditioning on any set $\mathbf{S} \subseteq \mathbf{B}(X) - \{Y\}$. Therefore all tests of step 2 will be positive and the algorithm will (correctly) include Y in $\mathbf{N}(X)$. In the case that X is not directly connected to Y, that means that Y is the parent of a number of children of X, say set $\mathbf{T}$, since it is a member of $\mathbf{B}(X)$. Therefore conditioning on a set that includes all parents of X and excludes $\mathbf{T}$ would block all paths from X to Y.

The proof of the second requirement is similar. The only difference is that instead of direct connectedness of two direct neighbors of X, say Y and Z, we are trying to ascertain connectedness through conditioning on any subset of $\mathbf{B}(Y)$ that necessarily includes X (without loss of generality we assume that $|\mathbf{B}(Y)| \leq |\mathbf{B}(Z)|$). If Y and Z are both parents of X, they would be dependent upon conditioning on any such set via the open path through X. If Y is a child of X, then conditioning on X blocks one of the paths to Z, and any other paths to Z are blocked by conditioning on the remaining parents of Y.

If the tests in step 3 are correct, step 4 will not attempt to eliminate any cycles. Otherwise it will remove a number of edges until there are no remaining remaining cycles. These edges can then be added in reverse direction without introducing cycles. The correctness of this relies on the same observation that that the last step does. We need to show that the incremental addition of edges for which the reverse direction necessarily imposes a cycle in the graph, does not itself introduce any cycles. The proof is by contradiction. The original graph, as the result of step 4, is acyclic. Assume that both $X \rightarrow Y$ and $Y \rightarrow X$ will produce a cycle when added to the set of edges (each individually) in an otherwise acyclic graph. Assume that adding $X \rightarrow Y$ will create cycle $X \rightarrow Y \rightarrow A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_l \rightarrow X$ and adding $Y \rightarrow X$ will create cycle $Y \rightarrow X \rightarrow B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_k \rightarrow Y$, as shown in the figure below.



However, that arrangement necessarily implies that the original graph already contained a cycle, namely $X \rightarrow B_1 \rightarrow \cdots \rightarrow B_k \rightarrow Y \rightarrow A_1 \rightarrow \cdots \rightarrow A_l \rightarrow X$, which is a contradiction. Therefore exactly one of $X \to Y$ and $Y \to X$ can be added to the graph without introducing a cycle and the next iteration still satisfies the inductive assumption that the graph is acyclic.

As mentioned above, the same argument may be used to show the correctness of the last step.

Timing Analysis

Step 1 involves $O(n^2)$ conditional independence (CI) tests. If $b = \max_X(|\mathbf{B}(X)|)$, step 2 does $O(nb2^b)$ CI tests. At worst $b = O(n)$, implying that the set of examples $\mathbf{D}$ was produced by a dense original Bayesian net. If we know that the upwards branching factor is less than or equal to u and the downwards branching factor less than equal to d, then $b \leq u + d + du$, which is a constant. Step 3 does $O(nb^2 2^b)$ CI tests, and the exponential factor may be limited similarly in the presence of branching factor information. Steps 4 and 5 do not do any independence tests. Checking for the existence of cycles in step 4 takes $O(m(n+m))$ time by employing a standard depth-first traversal of the graph for each existing edge. Step 5 is $O(m)$ in time. The last step can be implemented in a straightforward manner in $O(nb(n+m))$ time using depth-first traversal, which does not affect the asymptotic time of the algorithm.

The total number of CI tests for the entire algorithm is therefore $O(n^2 + nb^2 2^b)$ or $O(m^2 + nmb + (n^3 + n^2 b^2 2^b)\,|\mathbf{D}|)$ time. Under the assumption that b is bounded by a constant, this algorithm is $O(n^2)$ in the number of CI tests or $O(m^2 + n^3\,|\mathbf{D}|)$ time.

Discussion

The main advantage of the algorithm comes through the use of Markov blankets to restrict the size of the conditioning sets. The Markov blankets may be incorrect. The most likely potential error is to include too many nodes for reasons that are explained below. This emphasizes the importance of the "direct neighbors" step (step 2) which removes nodes that were incorrectly added during the Markov blanket computation step due to conditioning on large set of variables. The problem is the following: conditioning on n binary variables requires in 2^n dependency tests. Each such test compares two histograms, the histogram representing the probability distribution of binary variable X given the conditioning set $\mathbf{S}$ and the histogram representing the distribution of X given $\mathbf{S} \cup \{Y\}$. The two variables X and Y are pronounced dependent if for any of the $2^{|\mathbf{S}|+1}$ configurations of the set $\mathbf{S} \cup \{Y\}$, the corresponding histograms are "different enough." However, given a limited number of samples, if we are using the same set of samples for all tests, the probability that one of these tests will be positive and Y will be added to the blanket of X even if the two variables are not really dependent is increasing rapidly with increasing conditioning set size $|\mathbf{S} \cup \{Y\}|$. The "direct neighbors" step, which does a number of dependence tests between X and Y and declares them direct neighbors only if *all* these tests have high confidence, helps identify and correct these potential errors in the preceding Markov blanket phase.

5 Randomized Version of the GS Algorithm

The GS algorithm presented above is appropriate for situations where the maximum size of the Markov blanket of each variable under consideration is sufficiently small, since it depends on it through an exponential factor. While it is reasonable to assume that in many real-life problems this may be the case, certain ones, such as Bayesian image retrieval in computer vision, may employ finer representations. In these cases the variables used may depend in a direct manner on many others. For example, one may choose to use variables to characterize local texture in different parts of an image. If the resolution of the mapping from textures to variables is increasingly fine, direct dependencies among those variables may be plentiful, suggesting a dense underlying net. In these cases it may be prohibitively expensive to employ the exponential number of tests such as those done in the plain GS algorithm.

Another problem of the GS algorithm presented above, and one that has plagued independence-test based algorithms for Bayesian net structure induction in general, is that their decisions are based on a single or a few tests, making them prone to errors due to possible noise in the data. It would therefore be advantageous to allow multiple tests to influence a decision before determining the existence of an edge or its direction in the resulting net.

The following version of the GS algorithm addresses these two problems. First, by executing a fixed number of tests during the determination of the existence of an edge, using conditioning sets randomly drawn from the smallest of the two blankets, the time allocated to each edge may be finely controlled so that the algorithm is able to execute even for problems for which blanket sizes are large. An additional advantage of the approach is that it is now able to operate in an "anytime" manner, which is useful in real-time applications such as robot vision. Second, using Bayesian evidence accumulation, each structural decision depends only partly on the outcome of each single test (although the tests have to be appropriately weighted—see the discussion below).

This version of the algorithm is presented below. The algorithm computes the posterior probabilities that certain variables X and Y are direct neighbors and whether a possible link between them is directed as $Y \to X$ or $X \to Y$ after seeing a set of N data sets (denoted here as a vector of data sets) $\overrightarrow{\xi}_N$, that are assumed to be independently drawn. Each data set $\xi_i, i = 1, \ldots, N$ contains a number of examples.

1. [Compute Markov Blankets] (same as plain GS)
For all $X \in \mathbf{V}$, compute the Markov blanket $\mathbf{B}(X)$.

2. [Compute Graph Structure]
For each $X \in \mathbf{V}$ and $Y \in \mathbf{B}(X)$ do:

- Set $p \leftarrow \frac{1}{2}$.
- Set $\mathbf{T}$ to be the smaller of $\mathbf{B}(X) - \{Y\}$ and $\mathbf{B}(Y) - \{X\}$.
- Let $G \equiv G(X, Y) = 1 - (\frac{1}{2})^{|\mathbf{T}|}$.
- For each data set ξ_i, $i = 1, \ldots, N$, execute the following:
  - Set $\mathbf{S}$ to be a randomly chosen subset of $\mathbf{T}$.
  - Compute $d = P(X \leftrightarrow_{\mathbf{S}} Y \mid \xi_i)$.
  - Update the posterior probability p using the recursive formula

$$p \leftarrow \frac{pd}{pd + (1-p)(G + 1 - d)}$$

- Set $P(Y \in \mathbf{N}(X)) = P(X \in \mathbf{N}(Y)) = p$.
- Assign Y to be a member of $\mathbf{N}(X)$ and X to be in $\mathbf{N}(Y)$ if and only if $p > \frac{1}{2}$.

3. [Orient Edges]
For each $X \in \mathbf{V}$, $Y \in \mathbf{N}(X)$ do:

- Set $Q \leftarrow \frac{1}{2}$.

- Do for each $Z \in \mathbf{N}(X) - \mathbf{N}(Y) - \{Y\}$:
  - Set $q \leftarrow \frac{1}{2}$.
  - Set $\mathbf{U}$ to be the smaller of $\mathbf{B}(Y) - \{X, Z\}$ and $\mathbf{B}(Z) - \{X, Y\}$.
  - Let $G \equiv G(Y, Z) = 1 - (\frac{1}{2})^{|\mathbf{U}|}$.
  - For each data set ξ_i, $i = 1, \ldots, N$, execute the following loop:
    * Set $\mathbf{S}$ to be a randomly chosen subset of $\mathbf{U}$.
    * Compute $d = P(Y \leftrightarrow_{\mathbf{S} \cup \{X\}} Z \mid \xi_i)$.
    * Update the posterior probability q using the recursive formula

$$q \leftarrow \frac{qd}{qd + (1-q)(G + 1 - d)}$$

  - Update $Q \leftarrow \frac{Q(1-q)}{Q(1-q)+(1-Q)(1-G+q)}$.
- Set $P(Y \rightarrow X) = 1 - Q$.

For each $X \in \mathbf{V}$, $Y \in \mathbf{N}(X)$ do:

- Assign direction $Y \rightarrow X$ if $P(Y \rightarrow X) > P(X \rightarrow Y)$.
- Assign direction $X \rightarrow Y$ if $P(Y \rightarrow X) < P(X \rightarrow Y)$.
- Else assign the direction between X and Y randomly.

4. [Remove Cycles]
Do the following while there exist cycles in the graph:

- Compute the set of edges $\mathbf{C} = \{X \rightarrow Y$ such that $X \rightarrow Y$ is part of a cycle$\}$.
- Remove the edge $X \rightarrow Y$ in $\mathbf{C}$ that such that $P(X \in \mathbf{N}(Y))P(X \rightarrow Y)$ is minimum and put it in $\mathbf{R}$.

5. [Reverse Edges] (same as plain GS)
Insert each edge from $\mathbf{R}$ in the graph, reversed.

6. [Propagate Directions] (same as plain GS)
For all $X \in \mathbf{V}$ and $Y \in \mathbf{N}(X)$ such that neither $Y \rightarrow X$ nor $X \rightarrow Y$, execute the following rule until it no longer applies: If there exists a directed path from X to Y, orient $X \rightarrow Y$.

The derivations for the formula used above to compute the posterior probabilities is presented in the appendix. The same formula is used in two places in the algorithm, in updating the posterior probability of the existence of an edge and also for the corresponding probability on the direction of those edges that were determined to be present. The use of the parameter G, which acts to weigh each test, marks its difference from straightforward posterior application of formulas such as those found in [Pea88]. G roughly characterizes the connectivity of a node in terms of the size of its blanket, ranging from 0 (blanket size 1) to 1 (blanket size very large, tending to infinity). We can intuitively see its influence on the posterior probability by examining limiting cases for the parameters occurring in the formula, which is repeated below for the reader's convenience:

$$p \leftarrow \frac{pd}{pd + (1-p)(G + 1 - d)}.$$

We will examine the application of the formula to the existence of a direct link between two nodes X and Y. The application of the formula to the directionality of a link is similar. For simplicity we will assume that X has the smaller blanket of the two nodes, and therefore G is computed using $|\mathbf{B}(X)|$. We first look at the case when the dependence test $d = P(X \leftrightarrow_{\mathbf{S}} Y \mid \xi_i) = 0$. In this case, the posterior probability p becomes 0, as expected, since the hypothesis of a direct link between X and Y cannot possibly support this evidence. In the case that $d = 1$, the above formula becomes

$$p \leftarrow \frac{p}{p + (1-p)G}.$$

We look at two limiting cases for G. If $G = 0$ (*i.e.* $\mathbf{B}(X) = \{Y\}$), p becomes 1. This is reasonable since the high confidence of the dependence between X and Y combined with the absence of any other node in the blanket of X constitutes a direct indication of an edge between the two nodes. In case $G = 1$ (*i.e.* $|\mathbf{B}(X)| \to \infty$), a test indicating the dependence of X and Y conditioned on a random subset of $\mathbf{B}(X)$ is very weak evidence of a link between them, since the likelihood that a common ancestor will be absent or a common descendant will be present in the conditioning set (note that X is in $\mathbf{B}(Y)$ and vice versa) is high, and that would explain the evidence without the necessity of a link between the two nodes. Accordingly, the value of p does not change in this case.

The formulas in step 3 (Orient Edges) are seemingly complicated but they follow the same rationale. The quantity q that is computed in the loop is the probability that Y and Z, both direct neighbors of X, are dependent conditioned on a subset that contains X. However, the determination of whether $Y \to X$ or $Y \leftarrow X$ is an OR over all other direct neighbors Z. Our iterative formula that is based on multiple tests computes the probability that *all* tests are true *i.e.* an AND of tests. Therefore we have to use DeMorgan's law to convert the OR of tests into an AND of the negations of these tests. From this observation it is easy to that Q computes the posterior probability that $Y \not\to X$.

Finally, as a general note, we observe that the robustness of this version of the algorithm is at least as great as the one of the plain GS algorithm, given the same number of tests. The additional robustness comes from the use of multiple weighted tests, leaving for the end the "hard" decisions that involve a threshold (*i.e.* comparing the posterior probability with a threshold, which in our case is $\frac{1}{2}$).

6 Experimental Results

In this section we present results that demonstrate two main points. First, that both the plain and randomized GS algorithms perform significantly better than hill-climbing approaches in terms of the number of neighborhood and directionality errors, while maintaining a similar level of KL-divergence with respect to the original Bayesian nets from which the input data are drawn. And second, that the randomized version can execute much faster in cases of large neighborhood sized networks while maintaining good error rates compared to the plain GS algorithm.

Strictly speaking, the assumption made above and one that is used in the derivation of the formulas in the randomized version of the algorithm is that a new, independently drawn set of examples ξ_i should be used with each dependence test conducted. However, given that data is frequently expensive and/or scarce, in our experiments we use the same data in all tests, in order to demonstrate the performance of the algorithm under these adverse, but more realistic conditions. In the case that data is plentiful, the procedure would split the dataset into subsets of examples randomly, and use one subset for each dependence test.

Throughout the algorithms presented in this paper we employ standard chi-square (χ^2) conditional dependence tests (as is done also in [SGS93]) in order to compare the histograms $\widehat{P}(X)$ and $\widehat{P}(X \mid Y)$. The χ^2

test gives us the probability of error of assuming that the two variables are dependent when in fact they are not (type II error of a dependence test), from which we can easily derive the probability that X and Y are dependent. There is an implicit threshold τ involved in each dependence test, indicating how certain we wish to be about the correctness of the test without unduly rejecting dependent pairs, something that is always possible in reality due to the presence of noise. In our experiments we used 95% confidence tests, *i.e.* $\tau = 0.95$.

Reconstruction of an example network using different techniques is shown in Fig. 2.

We test the effectiveness of the algorithms through the following procedure: we generate a random rectangular net of specified dimensions and up/down branching factor. A number of examples are drawn from that net using logic sampling [Hen88] and are used as input to the algorithm under test. The resulting nets can be compared with the original ones along dimensions of KL-divergence and percent difference in edges and edge directionality. The definition of difference in the number of edges that is used is the number of edges which are not present or are extraneous in the induced net (as compared to the original one) divided by the total number of edges of the original network. This effectively uses a "Hamming distance" between the two nets, where a bit indicates the presence or absence of an edge from all possible edges. The difference in the directionality between the two nets is defined as the fraction of edges in the induced network that have the correct directionality among those whose presence was correctly determined.

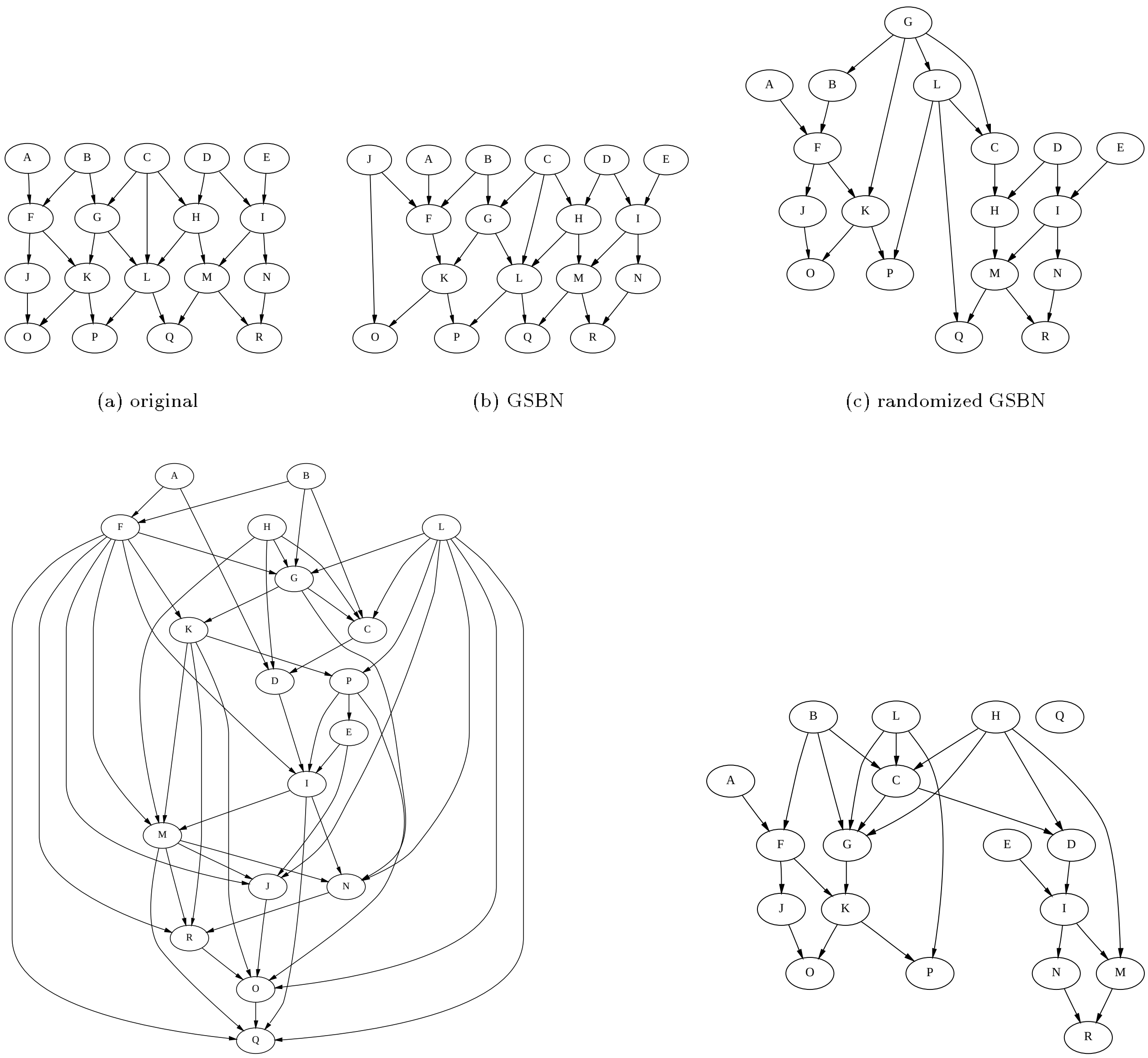
Fig. 3 shows the KL-divergence between the original and the reconstructed net as well as edge omissions/false additions/reversals as a function of number of samples used. 100 runs (Bayesian networks) were used for each data point. The figure demonstrates two main findings. First, that typical KL-divergences for both GS and the hill-climbing algorithm using the BIC criterion are similar and low, which shows good performance for applications where prediction is of prime concern. Second, the number of incorrect edges and the errors in the directionality of the edges present is much higher for the hill-climbing algorithms, making them unsuitable for accurate Bayesian net reconstruction. This may be of concern to certain applications (e.g. Data Mining). We also note that the KL-divergence of the randomized GS algorithm is the worst, it is still low and we expect that to become gradually lower as the number of randomized tests increases. As far as the number of edge and directionality errors it performs better than both hillclimbing algorithms.

Fig. 4 shows the effects of increasing the Markov blanket through an increasing branching factor. As expected, we see a dramatic (exponential) increase in execution time of the plain GS algorithm, though only a mild increase of the randomized version. The latter uses 100 conditional tests per decision, and its execution time increase is attributed to the (quadratic) increase in the number of decisions. Note that the edge percentages between the plain and the randomized version remain relatively close. The number of direction errors for the GS algorithm actually decreases due to the larger number of parents for each node (more "V" structures), which allows a greater number of opportunities to recover the directionality of an edge (using an increased number of tests).

7 Discussion and Related Work

Structural induction for Bayesian networks has been successful in restricted classes of graphs, such as trees and polytrees (trees with possibly multi-parent nodes). Optimal reconstruction of a Bayesian tree was shown by Chow and Liu [CL68]. Polytree reconstruction via a straightforward addition to the Chow-Liu method is presented in [RP89]. There are not many algorithms in the literature for general Bayesian net induction using independence tests. Two notable ones are the SGS and the PC algorithm, both presented in [SGS93][1].

[1] Steps that ensure that the resulting net will be legal (*i.e.* not contain any directed cycles) similar to steps 4 and 5 of the GSBN algorithm are not present in either the SGS or the PC algorithms, since they are unneccessary due to their assumption of perfect independence tests.



(a) original (b) GSBN (c) randomized GSBN

(d) hill-climbing (score: data likelihood) (e) hill-climbing (score: BIC)

Figure 2: An example reconstruction of a Bayesian network of 18 nodes and 25 edges by different structure induction techniques, from 15000 samples drawn from the distribution represented by the original network using logic sampling. The plain GS network contains a single directionality error and no neighborhood errors while the output of the randomized version (using 200 tests per decision) contains one neighborhood and 3 directionality errors. The hill-climbing approach using BIC as the scoring criterion contains 6 neighborhood and 4 directionality errors, while the data likelihood hill-climbing result contains 25 neighborhood and 5 directionality errors. (The definition of edge and directionality errors is in the text.) The KL-divergence of all four algorithms with respect to the original is approximately of the same order, around 2×10^{-5}.

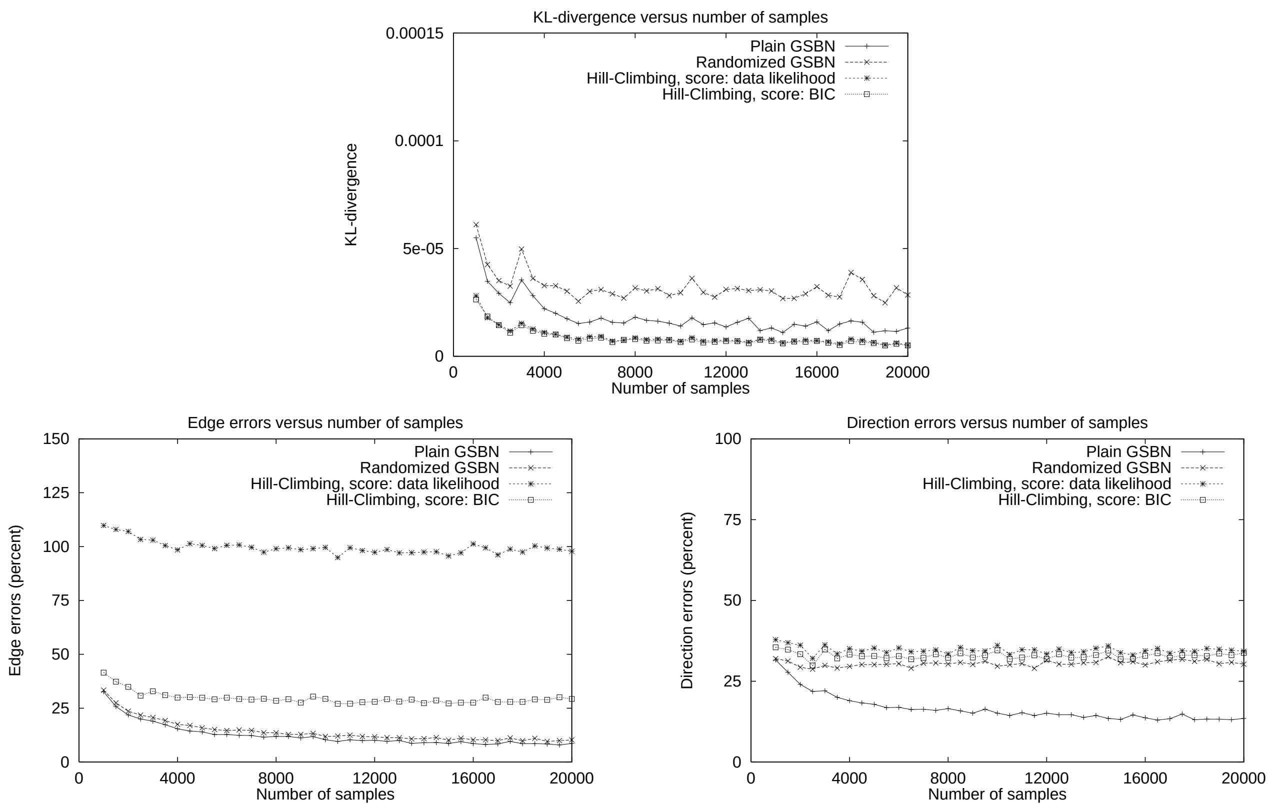


Figure 3: Results for a 5×5 rectangular net with branching factor 2 (in both directions, blanket size 8) as a function of the number of samples. On the top, KL-divergence is depicted for the plain GS, Randomized GS, and hill-climbing algorithms. On the bottom, the percentage of edge and direction errors are shown. Note that certain edge error rates for the hill-climbing algorithm exceed 100%.

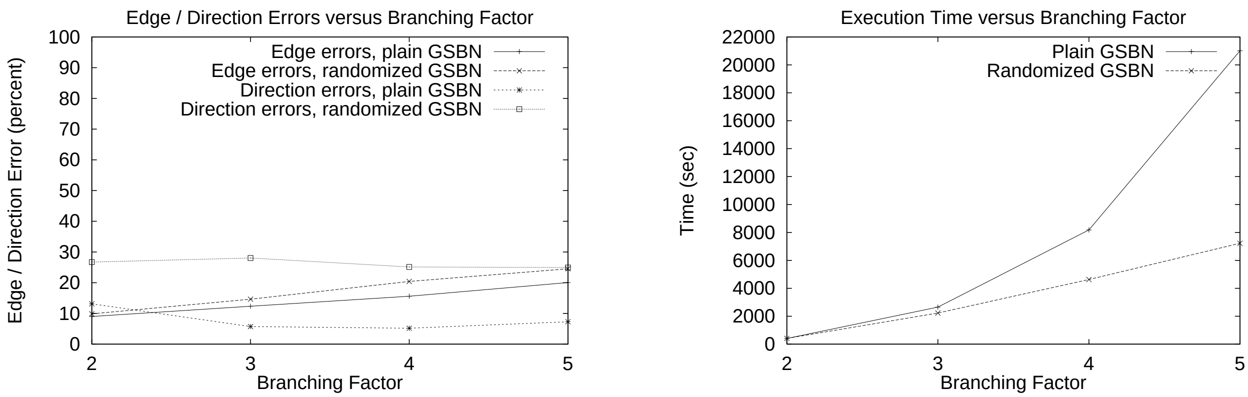


Figure 4: Results for a 5 ×5 rectangular net from which 10000 samples were generated and used for reconstruction, versus increasing branching factor. On the left, errors are slowly increasing (as expected) or remain relatively constant. On the right, corresponding execution times are shown. While the plain GS algorithm execution time increases exponentially with branching factor, the randomized version has a much milder increase in execution time.

The plain version of the GS algorithm presented here has similarities to both of them. It may be viewed as a version of the SGS algorithm with the conditioning sets restricted to the Markov blanket of each node, or PC without the linear probing for a separator it employs (see description of the PC algorithm below). The SGS algorithm proceeds in a fashion similar to the GS algorithm; for example, in order to determine the existence of an edge between two nodes X and Y, the SGS algorithm employs independence tests conditioned on every subset of $\mathbf{V} - \{X, Y\}$. This is clearly wasteful for sparse graphs where many variables may be irrelevant, and it also places unnecessarily requirements on the size of the data set that is needed for reliable structure discovery. The GS algorithm circumvents both problems by employing a preliminary step in which it discovers the local structure around each node. Concerning the recovery of the directionality of the links, the approach of the SGS algorithm is analogous to a global version of step 3 of the GS algorithm. It therefore suffers from the same comparative shortcomings as the ones mentioned above.

It is more fair, performance-wise, to compare the GS algorithm with an algorithm that heeds local structure such as the the PC algorithm [SGS93]. The PC algorithm is a much more efficient algorithm than SGS and it may be employed in practice on problems involving a significantly larger number of variables. In short, its operation is as follows. Initially a completely connected graph is constructed. This graph is then gradually "thinned" using conditional independence tests of increasing order (conditioning set size), starting from and empty set. For two variables X and Y connected with a hypothesized edge, the algorithm makes two tests, drawing the conditioning set of a certain size from the current set of direct neighbors of X in one test and the direct neighbors of Y on the other. In a sparsely connected original net, it is expected that many links will be eliminated early, reducing the sizes of the direct neighbors sets and improving the running times of subsequent steps. Each conditioning set that successfully separates two variables is recorded and used in a later phase where the directions of the remaining links are determined in a fashion similar to the SGS (and GS) algorithm.

[SGS93] present an informal worst-case analysis of the PC algorithm, which indicates that the number of independence tests used are $O(n^{k+1})$, where k is the maximum degree of a node of the original Bayesian net (in our notation, $k = u + d$, the sum of the maximum number of parents u and children d of any node). While both algorithms have a factor that is also exponential in k, which most likely is unavoidable, the factor in GS is 2^b, where $b = k + u$ in the worst case. However, although the PC algorithm is indeed efficient (*i.e.*

polynomial) under the assumption of a bounded neighborhood (k bounded by a constant), unfortunately the order of the polynomial depends on k, while the GS algorithm does not. For a given k (and therefore b), the GS algorithm has an asymptotic behavior with respect to n of $O(n^2)$, superior to the one achieved by PC algorithm. This is likely to make a difference in large graphs that are sparsely connected.

Other algorithms exist in the literature that do not make use of independence tests but take into account d-separation in order to discover structure from data. [CBL97] for example uses mutual information instead of conditional independence tests. The algorithm requires the ordering of the variables to be input to the algorithm.

There is another family of algorithms for structure recovery, as mentioned in the introduction. These algorithms typically employing heuristic search methods in the space of possible structures, using a number of scoring functions as the objective function to be optimized. Such scoring functions may be the Bayesian information criterion (BIC), mutual entropy [HC91], or the minimum description length [Suz96] (which is equivalent to the BIC [Hec95]). We will not expand on them here since they are not directly related to the algorithms proposed in this paper. However we will say that they are not guaranteed to construct a network that is faithful in terms of independence relations, nor one whose structure is "close" to the original one. Their guarantee is to return a network for which the scoring function, which is often related to the likelihood of the input data, is at a local maximum. (Such networks may be usefully employed to compute the probability of future data.) Therefore it is conceivable that an application of one of these algorithms be used to "fine-tune" the network returned by the GS algorithm, possibly producing a structure similar to the original one that is also close to the global maximum of the data likelihood.

8 Conclusion

In this paper we presented an efficient algorithm for computing the Markov blanket of a node and used it in two versions of the GS algorithm (plain and Monte Carlo) by exploiting the properties of the Markov blanket to facilitate fast reconstruction of the local neighborhood around each node, under assumptions of bounded neighborhood size. We also presented a Monte Carlo version that has the advantages of potentially faster execution speeds and added reconstruction robustness due to multiple tests and Bayesian accumulation of evidence. Simulation results demonstrate the reconstruction accuracy advantages of the algorithms presented here over hill-climbing methods. Additional results also show that the Monte Carlo version has a dramatical execution speed benefit over the plain one in cases where the assumption of bounded neighborhood may not hold, without significantly affecting the generalization error rate.

9 Acknowledgements

We would like to thank Avrim Blum for pointing out the close relation between *MaxLikelihoodDAG* and the Minimum Feedback Arc Set problem.

Appendix A: Computation of the probability $P(L \mid \overrightarrow{\xi}_n)$

The following derivation applies to the formulas in both steps 2 and 3 of the randomized version of the GS algorithm. In the derivation below the following symbols are used:

- $L(X,Y)$ represents the event that variables X and Y are permanently linked either through a direct connection (GS step 2) or by conditioning on a common descendant (step 3).
- $D_i(X,Y)$ for $i=1,\ldots,N$ are events that X and Y are dependent conditioned on a set of variables $\mathbf{S}_i$, a subset of $\mathbf{B}(X)-\{Y\}$ (that includes a possible common descendant in step 3).
- $\overrightarrow{\xi}_i = \langle \xi_1, \xi_2, \ldots, \xi_i \rangle$ is the first i data sets, represented as a vector of data sets.

In our notation we will omit the dependency on X and Y of the variables listed above for reasons of brevity. We assume that $X \in \mathbf{B}(Y)$ and $Y \in \mathbf{B}(X)$, as the probabilities that are referred to below are computed only for such variables in the randomized GS algorithm.

We are interested in computing the probability $P(L \mid \overrightarrow{\xi}_i)$ in terms of $P(D_i \mid \xi_i), i=1,\ldots,N$, the only direct evidence we may elicit from each data set ξ_i. In other words, the results of the tests D_i are the only information we use from ξ_i, making them in effect the sufficient statistics of the data set vector $\overrightarrow{\xi}_i$. This is one of our assumptions (a *sufficiency* assumption).

We will formulate $P(L \mid \overrightarrow{\xi}_i)$ in terms of $P(L \mid \xi_i)$, the probability of L given a single test only, and $P(L \mid \overrightarrow{\xi}_{i-1})$, the accumulated probability from previous data. We have

$$P(L \mid \overrightarrow{\xi}_i) = P(L \mid \xi_i \overrightarrow{\xi}_{i-1}) = \frac{P(\xi_i | L \overrightarrow{\xi}_{i-1}) P(L | \overrightarrow{\xi}_{i-1})}{P(\xi_i | \overrightarrow{\xi}_{i-1})}$$
$$P(\overline{L} \mid \overrightarrow{\xi}_i) = P(\overline{L} \mid \xi_i \overrightarrow{\xi}_{i-1}) = \frac{P(\xi_i | \overline{L} \overrightarrow{\xi}_{i-1}) P(\overline{L} | \overrightarrow{\xi}_{i-1})}{P(\xi_i | \overrightarrow{\xi}_{i-1})}$$

From the two equations above, and using the Markov assumption $P(\xi_i \mid L \overrightarrow{\xi}_{i-1}) = P(\xi_i \mid L)$ and $P(\xi_i \mid \overline{L} \overrightarrow{\xi}_{i-1}) = P(\xi_i \mid \overline{L})$, we get

$$\begin{aligned}
\frac{P(L | \overrightarrow{\xi}_i)}{1-P(L | \overrightarrow{\xi}_i)} &= \frac{P(\xi_i | L)}{P(\xi_i | \overline{L})} \frac{P(L | \overrightarrow{\xi}_{i-1})}{1-P(L | \overrightarrow{\xi}_{i-1})} \\
&= \frac{\frac{P(L | \xi_i) P(\xi_i)}{P(L)}}{\frac{P(\overline{L} | \xi_i) P(\xi_i)}{P(\overline{L})}} \frac{P(L | \overrightarrow{\xi}_{i-1})}{1-P(L | \overrightarrow{\xi}_{i-1})} \\
&= \frac{P(L | \xi_i)}{1-P(L | \xi_i)} \frac{1-P(L)}{P(L)} \frac{P(L | \overrightarrow{\xi}_{i-1})}{1-P(L | \overrightarrow{\xi}_{i-1})}
\end{aligned} \tag{1}$$

In the absence of any information, we assume that the probability that two variables are directly linked is equal to the probability that they are not. This implies $P(L) = \frac{1}{2}$. The only remaining term is $P(L \mid \xi_i)$:

$$P(L \mid \xi_i) = P(L \mid D_i \xi_i) P(D_i \mid \xi_i) + P(L \mid \overline{D}_i \xi_i) P(\overline{D}_i \mid \xi_i) \tag{2}$$

Since $P(L \mid \overline{D}_i \xi_i) = 0$ (if X and Y are not dependent they cannot possibly be linked), the second term at the sum above vanishes. For $P(L \mid D_i \xi_i)$, we use the sufficiency assumption stating that *knowledge* of the dependency between X and Y is all that we use to influence the probability of L. This allows us to infer $P(L \mid D_i \xi_i) = P(L \mid D_i)$. Therefore,

$$\left.\begin{aligned}
P(L \mid D_i \xi_i) = P(L \mid D_i) &= \frac{P(D_i | L) P(L)}{P(D_i)} \\
P(\overline{L} \mid D_i \xi_i) = 1 - P(L \mid D_i) &= \frac{P(D_i | \overline{L}) P(\overline{L})}{L(D_i)}
\end{aligned}\right\} \Rightarrow P(L \mid D_i) = \frac{P(D_i \mid L)}{P(D_i \mid \overline{L}) + P(D_i \mid L)} \tag{3}$$

If we know there is a direct link between X and Y, the probability they are dependent is 1:

$$P(D_i \mid L) = 1 \tag{4}$$

If however we know that they are not directly linked, and given that each belongs to the blanket of the other, this probability depends on whether we include all parents and no children of one of them in the conditioning set $\mathbf{S}_i$. For that purpose we introduce two new events, assuming (without loss of generality) that $|\mathbf{B}(X)| \leq |\mathbf{B}(Y)|$:

- $A_i(X)$ is the event that all parents of X are included in $\mathbf{S}_i$. We will abbreviate this as A in the formulas below.
- $C_i(X)$ is the event that at least one child of X is included in $\mathbf{S}_i$. This is abbreviated as C.

$$\begin{aligned} P(D_i \mid \overline{L}) \quad = \quad & P(D_i \mid AC\overline{L})P(AC \mid \overline{L}) + \\ & P(D_i \mid A\overline{C}\overline{L})P(A\overline{C} \mid \overline{L}) + \\ & P(D_i \mid \overline{A}C\overline{L})P(\overline{A}C \mid \overline{L}) + \\ & P(D_i \mid \overline{A}\overline{C}\overline{L})P(\overline{A}\overline{C} \mid \overline{L}) \end{aligned} \tag{5}$$

Given that there is no direct link between X and Y ($\overline{L} = 1$), the two variables are independent only in the case all parents and no children of X are included in the conditioning set, namely if $A\overline{C}\overline{L}$ is true. Therefore

$$\begin{aligned} P(D_i \mid AC\overline{L}) = 1 \\ P(D_i \mid A\overline{C}\overline{L}) = 0 \\ P(D_i \mid \overline{A}C\overline{L}) = 1 \\ P(D_i \mid \overline{A}\overline{C}\overline{L}) = 1 \end{aligned} \tag{6}$$

Since the algorithm picks the members of the conditioning set entirely randomly we have

$$\begin{aligned} P(AC \mid \overline{L}) = P(A)P(C) = \quad & \tfrac{1}{2^\alpha}(1 - \tfrac{1}{2^\beta}) \\ P(A\overline{C} \mid \overline{L}) = P(A)P(\overline{C}) = \quad & \tfrac{1}{2^\alpha}\tfrac{1}{2^\beta} \\ P(\overline{A}C \mid \overline{L}) = P(\overline{A})P(C) = \quad & (1 - \tfrac{1}{2^\alpha})(1 - \tfrac{1}{2^\beta}) \\ P(\overline{A}\overline{C} \mid \overline{L}) = P(\overline{A})P(\overline{C}) = \quad & (1 - \tfrac{1}{2^\alpha})\tfrac{1}{2^\beta} \end{aligned} \tag{7}$$

where $\alpha(X) \equiv \alpha$ is the number of parents and $\beta(X) \equiv \beta$ the number of children of X and Y. Combining Eq. (5) with Eqs. (6) and (7) we get

$$P(D_i \mid \overline{L}) = 1 - \frac{1}{2^{\alpha+\beta}} \tag{8}$$

Since α and β are not known in advance, we can approximate (overestimate) their sum with $|\mathbf{B}(X)| - 1 \equiv |\mathbf{B}| - 1$ (we subtract 1 because Y is also a member of $\mathbf{B}$):

$$P(D_i \mid \overline{L}) \approx 1 - \frac{1}{2^{|\mathbf{B}|-1}} \equiv G \tag{9}$$

G is a measure of how connected a node (in this case, X) is to the remaining variables. Its value ranges from 0 to 1 as $|\mathbf{B}|$ takes values from 1 to ∞. Combining Eqs. (2), (3), (4) and (9) (where we assume equality from now on), we get

$$P(L \mid \xi_i) = \frac{P(D_i \mid \xi_i)}{1+G} \tag{10}$$

This equation has an easily seen and intuitive appealing interpretation: when $|\mathbf{B}|$ is large ($G = 1$), the evidence of a test indicating dependence between X and Y conditioned on a randomly drawn subset $\mathbf{S}_i$ of the blanket of X bears little evidence of a direct link between them, since the probability that it includes all parents and no children of X is small and therefore the posterior probability of L is close to $\frac{1}{2}$. When $|\mathbf{B}|$ is 1 ($G = 0$), however, the evidence for L is strong and the above probability becomes 1, since, in the absence of any common parents or children of X (or Y), any test indicating dependence is direct evidence of a link between X and Y.

Combining Eq. (1) with Eq. (10) and solving for $P(L \mid \overrightarrow{\xi}_i)$, we obtain the final recursive formula:

$$\boxed{P(L \mid \overrightarrow{\xi}_i) = \frac{P(L|\overrightarrow{\xi}_{i-1})\, P(D_i|\xi_i)}{P(L|\overrightarrow{\xi}_{i-1})\, P(D_i|\xi_i)+(1-P(L|\overrightarrow{\xi}_{i-1})\, (G+1-P(D_i|\xi_i))}}$$

Appendix B: *MaxLikelihoodDAG* is NP-complete

Theorem Given a directed graph $G(\mathbf{V}, \mathbf{E}, w)$ such that $\forall X, Y \in \mathbf{V}$, if $(X, Y) \in \mathbf{E}$ then $(Y, X) \in \mathbf{E}$, where $w(\cdot \to \cdot)$ is a weight function for each edge direction, the problem of determining the acyclic graph of the maximum product of edge weights (call it *MaxLikelihoodDAG*) is NP-complete.

Proof We reduce from the minimum feedback arc set problem (*MFAS*). The *MFAS* problem is about finding the smallest set of edges in a directed graph $G_{MFAS}(\mathbf{V}_{MFAS}, \mathbf{E}_{MFAS})$ whose removal will make the resulting graph undirected. To reduce it to *MaxLikelihoodDAG* we define $G(\mathbf{V}, \mathbf{E}, w)$ as follows

$$\mathbf{V} = \mathbf{V}_{MFAS}$$

$$\mathbf{E} = \mathbf{E}_{MFAS} \cup \{(Y, X) \mid (X, Y) \in \mathbf{E}_{MFAS}\}$$

$$w(X \to Y) = \begin{cases} 2 & \text{if } (X, Y) \in \mathbf{E}_{MFAS} \\ 1 & \text{if } (X, Y) \notin \mathbf{E}_{MFAS} \end{cases}$$

Calling *MaxLikelihoodDAG* will return a subgraph such that $\prod_{(X,Y)\in\mathbf{E}} w(X \to Y)$ is maximum or, equivalently, $\sum_{(X,Y)\in\mathbf{E}} \log_2 w(X \to Y)$ is maximum. Removing the edges that have $\log_2 w(X \to Y) = 0$ gives us a solution to the *MFAS* problem. This is because of two reasons. Firstly, since removing edges cannot introduce a cycle and since all remaining edges in the solution are members of $\mathbf{E}_{MFAS}$, it is a legal solution to the feedback arc problem. Secondly, because it contains a maximum number of edges it is a solution to the minimum feedback arc problem. We prove this claim by contradiction: suppose that there exists an acyclic subgraph of G_{MFAS} that has a greater number of edges. Then by using the edge weight assignment described above, we can produce a better solution (greater weight product) to *MaxLikelihoodDAG*, which is a contradiction.

This proves NP-hardness of *MaxLikelihoodDAG*. NP-completeness is due to the polynomial-time conversion (in the size of the graph) from the *MaxLikelihoodDAG* solution to a *MFAS* one, which simply involves the omission of the edge weights.

References

[Ago90] John Mark Agosta. The structure of Bayes networks for visual recognition. In *Uncertainty in Artificial Intelligence*, pages 397–405. Elsevier Science Publishers B.V. (North-Holland), 1990.

[CBL97] J. Cheng, D. A. Bell, and W. Liu. An algorithm for Bayesian network construction from data. In *Artificial Intelligence and Statistics*, 1997.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[HC91] E. Herskovits and G. Cooper. Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 117–125. Elsevier Science Publishers B.V. (North-Holland), 1991.

[Hec95] D. Heckerman. A tutorial on learning bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, March 1995.

[Hen88] M. Henrion. Propagation of uncertainty by probabilistic logic sampling in Bayes' networks. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence 2*. Elsevier Science Publishers B.V. (North-Holland), 1988.

[J85̈] M. Jünger. *Polyhedral combinatorics and the acyclic subdigraph problem.* Heldermann, Berlin, 1985.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[RP89] G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 222–228. Elsevier Science Publishers B.V. (North-Holland), 1989.

[SGS93] P. Spirtes, G. Glymour, and R. Scheines. *Causation, Prediction, and Search.* Springer-Verlag, New York, 1993.

[Suz96] J. Suzuki. Learning Bayesian belief networks based on the MDL principle: an efficient algorithm using the branch and bound technique. In *Proceedings of the International Conference on Machine Learning*, Bally, Italy, 1996.

[VP90] T. S. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Uncertainty in Artificial Intelligence*, pages 220–227. Elsevier Science Publishers B.V. (North-Holland), San Francisco, 1990.