A HYBRID APPROACH TO LEARNING BAYESIAN NETWORKS WITH DEPENDENCY CONSTRAINTS

by

Gustavo Frigo

B.Sc., Simon Fraser University, 2007

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

APPROVAL

Name:	Gustavo Frigo
Degree:	Master of Sciences
Title of Thesis:	A Hybrid Approach to Learning Bayesian Networks with Dependency Constraints

Examining Committee: Dr. Ted Kirkpatrick
Chair

____

Dr. Oliver Schulte, Associate Professor, Computing Science
Simon Fraser University
Senior Supervisor

____

Dr. Martin Ester, Professor,Computing Science
Simon Fraser University,
Supervisor

____

Dr. Greg Mori, External Examiner,
Assistant Professor, Computing Science
Simon Fraser University

Date Approved: JUL 10, 2009

Abstract

Bayesian networks present a useful tool for displaying correlations between several variables. This thesis presents a hybrid search strategy for structure learning in Bayesian networks whose structure is a directed acyclic graph. The general strategy performs a local search that meets the following criteria:

1. The Markov blankets in the model should be consistent with dependency information from statistical tests.
2. Minimizes the number of edges subject to the first constraint.
3. Maximizes a given score function subject to those constraints.

The strategy is adapted and optimized for learning structures for both discrete and continuous networks. Both algorithms are discussed and tested empirically both on synthetically generated structures, and on real networks. We show that adding dependency constraints can improve the quality of the learned models. Furthermore, unlike purely structural strategies, our hybrid method is robust enough to output high quality models even when available data is sparse.

Contents

List of Figures

Chapter 1

Introduction & Notation

1.1 Overview

Bayesian networks (BN) have been around since the late 80's, when a major publication on the subject appeared in 1988 [21]. Many applications have been built using BN's (including the infamous paper clip aid used in Microsoft Word © 97). A BN is a tool that can be used to model uncertainty arising in different types of data. Formally a BN consists of a directed acyclic graph G, together with parameters attached to each vertex in G, that model the underlying distribution. Each vertex represents a single attribute (or in statistical terms, a random variable) to be modeled. Such vertices may take either discrete or continuous values depending on the type of the attribute. The vast majority of BN's are either purely discrete (that is, every attribute takes on a finite number of values), or purely continuous [16]. The edges of G represent (among other things) statistical dependencies between the variables.

In a discrete BN (that is, a network in which every node takes on discrete values), every vertex has a conditional probability table (CPT) associated to it. The power of these networks lies in the fact that they can efficiently encode joint probability tables for a given joint distribution. Each vertex v in G contains conditional probabilities which only depend on the parents of v (the parents of v being all nodes which have an edge pointing into v). Each of the internal tables grows exponentially only in the number of parents of a particular node. Since the number of parents a particular vertex has is typically not very large, the amount of storage required to fully encode a BN is reasonably small by today's standards. On the other hand, encoding a CPT for the full joint distribution grows exponentially large with the number of nodes, thus a BN will usually save a substantial amount of space.

In contrast, continuous networks (called structural equation models or SEM) do not encode CPT's for every node; rather, a parameterized distribution is assigned to each vertex in the graph. This distribution is typically assumed to be a Gaussian distribution with parameters μ for the mean, and σ for the standard deviation. For example, the income of a certain business could be represented as a Gaussian distribution; intuitively, the parameter μ would represent the expected income of the business, while σ would represent a measure of the fluctuation of such income.

Given a BN, one can make probabilistic queries to the network. One way to query a BN is through the direct use of statistical formulas for conditional and joint probabilities. Two of the most used formulas are: the definition of a joint probability in terms of its conditional probability

$$P(A\text{and}B) = p(A) * P(A|B) = P(B) * P(B|A).$$

and the law of total probability ([11], pp 81)

$$P(B) = \sum_{i=1}^{k} P(B|A_i P(A_i$$

The process of making statistical queries to a BN is called inference. In theory, retrieving an arbitrary probability has been shown to be NP hard ([20] section 3.6); however, in practice, many interesting probability facts can be retrieved in a small amount of time. In cases in which inference becomes infeasible, one can use approximation algorithms which estimate the desired values in an efficient manner.

By assuming that every variable follows a Gaussian distribution, the inference process for continuous networks simplifies as it can be shown that a linear combination of Gaussian distributions results in another Gaussian distribution ([20] chapter 4). Note that in the case of a discrete BN, the inference process will return an exact value for a probability query (for example, the probability P of an event A is 35%). In a continuous BN, the inference algorithm will return two numbers representing the parameters of a Gaussian distribution (that is, the mean and variance). In this case, instead of having an exact number for a given event, we have a model (the Gaussian distribution) of the behaviour of the random variable. In order to obtain the probability for a particular range of values for this random variable, one can simply use a standard method for inference of a Gaussian distribution (such as the one described in [11] chapter 4, section 3).

In general, BN's are used to graphically represent interdependencies among several variables. Given a particular dataset, one can use a number of learning algorithms for constructing BN's in order to visually examine correlations between variables. Since BN's also encode probabilistic information, they can also be used as predictive models. An arbitrary variable can be chosen as the predictor variable, and the probabilities associated to events for this particular variable can be updated (via inference) as more evidence becomes available. Interestingly enough, if an user is only interested in predicting outcomes for a single event (modeled by a single variable), then a full generative model may not be necessary. Naive Bayes nets are a common model used for prediction. These types of BN have a fixed structure (that is, the graph corresponding to the BN is predetermined) and thus are not suitable for examining correlations. Even though naive BN's have a simple predetermined structure, they have been shown to be useful for the prediction task [28] . In a BN, the joint distribution is equal to the product of the conditional distributions of all nodes given values of their parents in a DAG ([20], section 1.3.1). In symbols, we have

$$P(v_1 = a_1, \ldots, v_n = a_n) = \prod_{i=1}^{n} P(v_i = a_i | \mathit{parents}_i)$$

where $P(v_i = a_i|\mathit{parents}_i)$ is the conditional probability that child node v_i takes on value a_i, given that its parents have the values $\mathit{parents}_i$) specified by the assignment $v_1 = a_1, \ldots, v_n = a_n$. The conditional probability is one of the parameters of the Bayes net specified in the CP-table for node v_i.

This factorization can be used in the inference process to calculate the probability (conditional or otherwise) of any node in a BN.

1.1.1 Illustrative Example

It will be useful at this point to go over a small BN in order to illustrate some of the concepts introduced in the previous section as well as to show how a BN can be used in practice. The fire network (taken from [9]) will serve as a good example. This network consists of 6 vertices connected with seven edges (see fig 1.1).

This network models events related to a fire alarm. The network can be interpreted as follows: Having a fire directly affects the probability of seeing smoke as well as the probability of a fire alarm being triggered. The triggering of a fire alarm is also affected by having the fire alarm tampered with. A fire alarm will affect whether people leave the

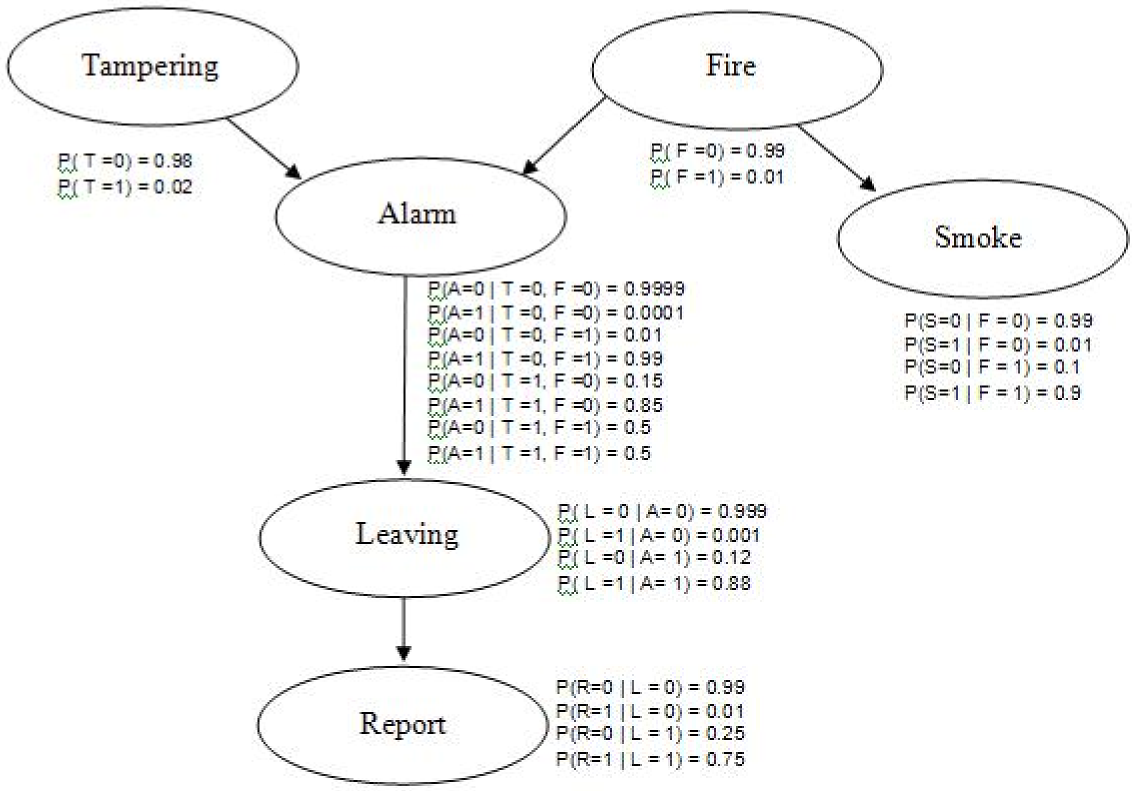


Figure 1.1: The fire alarm network

building, and this in turn will affect if a report is written. In this case the network is discrete. Each variable takes on two values: 0 and 1 (although in the general case variables may take on more than 2 values). For example, the variable fire takes the values 1 and 0. In the former case this is interpreted as the absence of a fire, while in the later this would be interpreted as a fire being present.

This only describes the structural part of a BN. In order for the model to be complete, a CPT must be associated to each node. These tables are also shown in figure 1.1. For example, for the variable fire, the CPT has 2 entries corresponding to marginal probabilities of a fire being present or absent (in this case, a fire is present with probability 1% and absent with probability 99%). In this case, we can also see that the size of any CPT is proportional only to the parents of a variable, as we only record probabilities of a variable conditional only on its parents.

Now that the model is completely described, one may want to issue some probabilistic queries. For example, we may want to know what the probability of people leaving the building is. As a first step, one must look at the CPT of the leaving node. Here we have that P(leaving=1 given alarm=0) = 0.001 and that P(leaving=1 given alarm=1) = 0.88 (indicating that people are much more likely to leave the building given that a fire alarm is heard). Using various statistical formulas, we conclude that:

$$P(leaving = 1) = P(L = 1|A = 0) * P(A = 0) + P(L = 1|A = 1) * P(A = 1)$$

This results in two subqueries to the network: P(alarm=0) and P(alarm=1). By recursively solving these subqueries, one can obtain the desired result P(leaving=1) = 0.0234 (about 2%). It is not hard to imagine that using a simple recursive algorithm like the one used for the example can be quite time consuming for a large BN. In some cases it is possible to reduce some of the inference steps; however, as mentioned before, there is no known inference algorithm that runs in polynomial time.

1.1.2 Usefulness

Bayesian networks have been used in several disciplines, ranging from medical applications to asses the status of patients, to financial domains to determine credit scores for individuals [24]. In fact, one of the most famous networks used as a benchmark for the performance of many learning algorithms, is the alarm network [2]. This net is used as a medical diagnostic system and consists of a directed graph with 37 nodes (each of which takes on 2,3,or 4 discrete values). Even disciplines from social sciences such as economics, and sociology utilize Bayesian networks to model certain aspects of human behaviour. In [14], an example of a continuous BN (an SEM) is used to model sociological shifts in populations. The fact that Bayesian networks are used in such a wide range of domains serves as an indicator of their usefulness. In the rest of the chapter, I will describe some of the advantages as well as the limitations of using (as well as learning) BN's. Having established these issues, I will describe the novel contributions made during my research.

1.2 Learning Algorithms: State of the art and Open Issues

Learning a BN consists of two parts: structure learning, and parameter estimation. In structure learning, one is interested in finding a structure which represents many of the correlations found in the underlying distribution. Parameter estimation parametrizes the structure so that the BN accurately models the distribution. If every variable can be observed in the data (that is, every variable in the BN is also present in the data) one may choose a maximum likelihood (ML) type of learner in order to do parameter estimation. In a ML paradigm, the structural part of the BN is parametrized in such a way as to maximize the likelihood that the data was generated by the model. For Example, in a discrete BN, ML consists of individually maximizing the likelihood that each of the CP tables generates the data (this in turn corresponds performing a frequency count on the data, and returning these relative frequencies as parameters). If the BN has latent variables (that is, a variable which appears in the BN, but is not observed in the data), then other more sophisticated methods such as expectation maximization must be employed in order to perform parameter estimation.

There are many learning algorithms available for constructing the structural portion of a BN from data. In general, structure learning algorithms are designed using two radically distinct paradigms: score based, and constraint based learning. The design of a score based learner consists of a greedy local hill climbing search combined with a score function; on the other hand, a constraint based learner typically uses a statistic test (such as Pearson's correlation coefficient, or χ^2 testing) to test whether two attributes are independent or not. This information can then be used to constraint the structure of the network.

Scoring based algorithms typically use a greedy procedure to find networks that (locally) optimize a given score function. The reason for this is that the number of possible networks grows exponentially with the number of attributes. This makes any type of exhaustive search infeasible for large graphs. A famous algorithm for score based learning is the greedy equivalence search algorithm (GES) described in [7]. GES describes a general hill climbing strategy for finding locally optimum networks given a score function (examples of such functions will be described in detail later on). Chickering proves that under reasonable assumptions, "GES correctly identifies the optimal solution [as the number of data instances goes to infinity]" [7]. This result proves that given enough data, GES paired with a suitable scoring function will indeed find the globally optimum Bayesian network.

Constraint based learning algorithms normally make use of independence relations among variables. The way in which such models are constructed is described in detail in chapter 5 of [25]. Typically, a backwards search strategy is employed; that is, if a network consists of n vertices (corresponding to n attributes), constraint based strategies build **K**n (The complete network on n vertices) as a starting point for searching. If some statistical test determines that two attributes happen to be independent, then the corresponding vertices must not be adjacent in the resulting Bayesian network. These direct constraints as well as other more subtle ones constitute the core of what is implemented in search algorithms such as PC [20] p.550 and CPC [22]. A few other constraint based algorithms are described in chapter 5 of [25], all of which come in different variants.

1.2.1 Limitations

Constraint based methods and score based methods perform well in terms of computational time required. In theory these algorithms do not have polynomial upper bound on the number of steps required to finish (see [20] p.552 and [8] for further details). In practice, the running time performance is quite good; and for this reason, it is usually not considered to be a significant limitation. Models produced by such algorithms are usually measured using performance metrics which assess the degree to which the underlying data has been learned. Such metrics compute how well the model represents this data and how complex the model is. A combination of these measures is put together in order to evaluate the overall quality of the constructed network.

The main limitation in constraint based methods arises from the fact that usually such methods use independencies among variables in order to restrict the target structure. The advantage of this approach is that independence facts are easily mapped into structural constraints; for example, if two variables A and B are known to be independent, then they must not appear as adjacent nodes in the corresponding Bayesian network. The problem with this is that there are no statistical methods that can reliably test whether two variables are independent or not. There are two types of errors associated with this kind of statistical testing: Type I errors and type II errors.

A type I error (or false positive) "consists of rejecting the null hypothesis H0 when it's true" [11] p319. In other words, a type I error occurs when a statistical test deems two variables to be not independent even though they are independent in the true distribution. A type II error (or false negative) "involves not rejecting H0 when H0 is false" [11] p319. Translated into the setting of independence testing, a type II error occurs when a statistical test entails that two variables are independent, when in fact they are not independent in the true distribution.

In general, when using statistical independence as a criteria for structure learning, the resulting graphs will be very sparse due to the fact that the number of type II errors can grow considerably large. Given that such methods may query a relatively large number of independence tests (proportional to a polynomial factor of the number of variables), it is easily seen how the accumulation of errors can lead to a model which is far from the true model.

On the other hand, score based methods do not suffer from this phenomenon as they do not rely on a statistical test in order to produce a model. The main restriction of a score based learner is that given a small sample of data, it may fail to discover structurally significant correlations. In fact, empirical studies have shown that the GES strategy with certain score functions consistently underfit data. One possible explanation for this is that some of the structural properties, which may be evident in the data, are not captured by the model because it would make it too complex. In other words, when the data is sparse, the score function acts conservatively choosing simple models which may underfit the structure of the true distribution.

1.3 Novel contributions

in the previous section (1.2.1) it was established that both constraint based, as well as scoring based algorithms, have shortcomings. Ideally a learner would use both the information given by a certain score function, as well as the structural constraints imposed by the dependency relationships among attributes. This is what the new algorithm (I-map) accomplishes.

I-map is a hybrid algorithm that combines both a hill climbing search strategy (together with a score function) as well as structural constraint information given by statistical tools. The idea behind the I-map algorithm is to use the search strategy defined by GES, together with a score function in order to guide the search for an optimal network; during the search process, I-map will also use dependency constraints in order to ensure that the resulting network satisfies certain structural properties (defined later on).

An important design decision should be pointed out: I-map uses dependency constraints rather that independency constraints. The reason for this is that, as mentioned before, is well know in the statistical community that testing for independence relationships (using something like χ^2) is not reliable for small sample sizes ([13] chapter 5); however, dependency testing is much more reliable in the following sense: Given that a statistical test finds a dependency between two variables, there is a high chance that the dependency is indeed part of the underlying data. That is, the type I error is not too large. In this sense, the

I-map algorithm differs from other structure learning algorithms since it uses a relatively small set of reliable dependencies in order to constraint the search space. The score can then be used to bridge the gap between the true model, and what would be learned by a purely structural algorithm.

Chapter 2

Definitions

This chapter will introduce some general definitions which will be used in later chapters.

2.1 Graph Theory Background

This section will introduce some basic definitions developed from graph theory. These definitions will serve as the base of the theory developed in later chapters. As mentioned in the previous chapter, the main component of a BN is a directed acyclic graph (DAG). a DAG consists of a set **V** of vertices together with a set **E** of (directed) edges connecting the vertices in **V** . An edge e which goes from vertex A to vertex B is often denoted by $A \rightarrow B$. Two nodes in **V** are adjacent if there is some edge in **E** which goes from A to B or from B to A. Since we are dealing with directed graphs, it is important to note that there is a difference between having an edge going from A to B, and having an edge from B to A. Undirected graphs on the other hand make no such distinction. There are graphical models other than BN which do use undirected graphs as the underlying structure. These methods however fall outside of the scope of this thesis. (for a reference on such methods, see [13]).

2.1.1 Node relationships: parents, children and spouses

It is sometimes useful to know what surrounds a particular node since in a BN structure, those vertices which surround a particular node directly affect it. The neighbourhood of a vertex U (notation **N** (U)) is defined as the set of nodes in **V** which are adjacent to U. There are two types of nodes which can appear in the neighbourhood of a vertex: parents

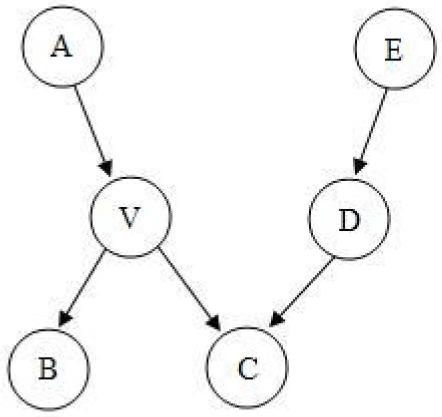


Figure 2.1: A simple DAG with 6 vertices and 5 edges. In this case, the vertex V has 3 neighbours in its neighbourhood (**N** (V)={A,B,C}). The nodes B and C are the children of V, while the node A is the only parent of V. The only two nodes which have a spousal relationship are nodes V and D.

and children. The parent nodes of a vertex U are those vertices which have edges going into U (intuitively these are the edges that are "just behind" U). Similarly, W is a child of U if there is an edge from U into W (so child nodes are those nodes "just ahead" of U). Figure 2.1 illustrates these concepts.

There is one other important relationship between vertices. A vertex V is a spouse of the vertex U if they have at least one child vertex in common. For example, in figure 2.1, the nodes V and D are spouses of each other as they have a child in common (vertex C).

2.1.2 Paths, Directed Paths and Acyclicity

There is one property of a DAG that has not yet been described: acyclicity. In order to understand acyclicity, it is useful to introduce the concept of a simple directed path (or a directed path) and that of a cycle. A directed path from vertex V to vertex U is defined as a sequence S of vertices in **V** such that for any two adjacent vertices V1,V2 in S, the edge $V1 \rightarrow V2$ is in **E** . A directed path is simple just in case every adjacent pair of vertices in S is unique (in other words, every edge in the path is traversed once). Unless stated otherwise, a simple path will be called a path. If there is a directed path from a node V to a node U, then U is called a descendant of V, and V is a predecessor of U.

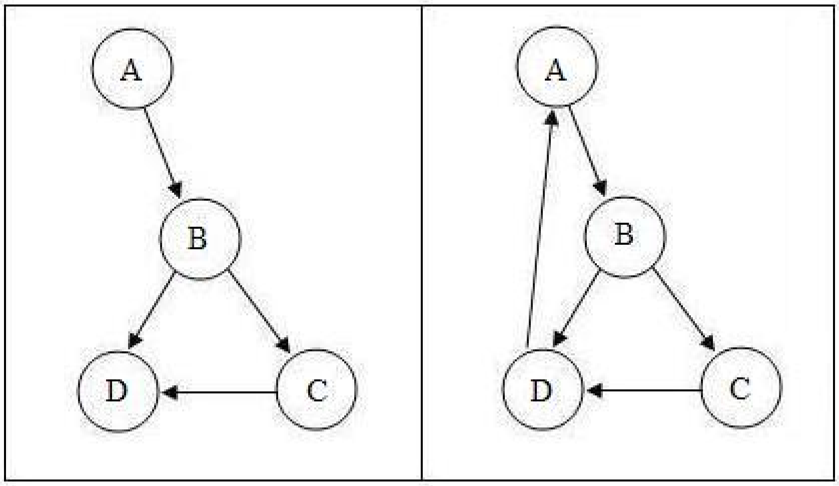


Figure 2.2: Left: an example of a DAG. Note that there are two directed paths from A to D: {A,B,D} and {A,B,C,D}). Right: an example of a graph which is not acyclic. This graph contains two cycles: {A,B,D,A} and {A,B,C,D,A}.

A cycle is a directed path which starts and ends in the same vertex. An acyclic graph is a graph which contains no cycles (that is, one cannot find a sequence of nodes which start and end in the same vertex. As a special case of this rule, a DAG cannot contain both an edge $U \rightarrow V$ and an edge $V \rightarrow U$ (since this constitutes a cycle of 2 vertices). Figure 2.2 gives examples of directed paths and cycles.

A simple (undirected) path from vertex U to V is a sequence S of vertices such that for any two adjacent vertices V1,V2 in S, either the edge $V1 \rightarrow V2$ is in $\mathbf{E}$, or the edge $V2 \rightarrow V1$ is. Essentially an undirected path is a path in which the edge direction is not taken into account. For example, in figure 2.2, the left graph has undirected paths from D to A. Any path has internal nodes which are those nodes that are not situated at the endpoints of the path. In an undirected path these internal nodes may have different types of edges adjacent to it. For example, let V be an internal node of a path P, then V may have one of the following edge configurations: (1) $A \rightarrow V \rightarrow B$, (2) $A \leftarrow V \rightarrow B$, or (3) $A \rightarrow V \leftarrow B$. When an internal node V has two incoming edges (this corresponds to the configuration $A \rightarrow V \leftarrow B$), V is called a collider of the path P.

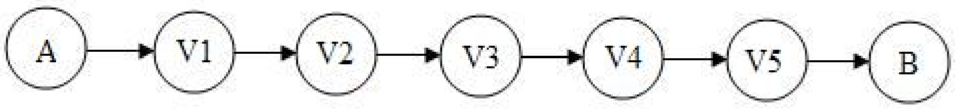


Figure 2.3: A graph consisting of a single path from vertices A to B

2.2 Dependencies and their interaction with Bayesian Networks

Statistical dependencies are situated at the core of BN. Recall from statistics that two random variables X and Y are independent if P(X,Y) = P(X)P(Y), (where P(Z) denotes the probability that the random variable Z takes on value z, for every value). As well as dependencies, one is often interested in finding conditional (in)dependencies between vertices in a BN. In a distribution D, two random variables X and Y are conditionally independent given a set of random variables Z if $P(X,Y|Z) = P(X|Z)P(Y|Z)$ (Given that P(Z) ¿ 0). If variables X and Y are independent given a third set of variables Z, the notation used is $(X \perp\!\!\!\perp Y|Z)_D$. Similarly, if X is dependent on Y given Z, the notation becomes $(X \not\perp\!\!\!\perp Y|Z)_D$

When working with a BN, it is useful to characterize when two variables are dependent from one another. If no additional information is known, then it is usually the case that many variables will be dependent among each other. For example, if the underlying graph of a BN is a single (long) path from a node A to a node B (passing through vertices V1,V2,V3, and V4) then every variable is dependent on each other (see figure 2.3).

It is in fact more useful to measure the degree of dependence among variables. In order to do this, one can resort to conditional independence. The structural part of any BN (that is the graph) defines a separability relationship among vertices named d-separation [21, Ch.3.3]. Two nodes U and V are d-separated given a set of nodes S if along every (undirected) path P between U and V there is a node W satisfying one of the following conditions:

1. W is a collider on P and none of W nor any of its descendants are in S, or
2. W is not a collider on P and W is in S.

If two vertices U and V are d-separated given a set S, then the corresponding random variables U and V are independent given S; if u and v are d-separated given S in G, it is typically written as $(U \perp V|S)_G$. Figure 2.4 illustrates the concept of d-separation.

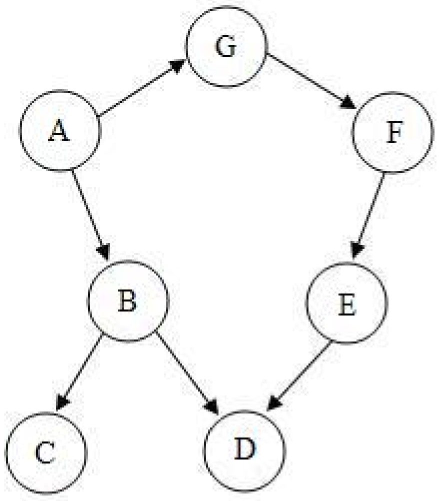


Figure 2.4: Example of a DAG in which the node A is d-separated from F given G (in other words, $A \perp G | \{G\}$). There are two paths from A to F. One of these paths contains the collider node D. The other path has no colliders, but contains the vertex G. Therefore the only node needed in the conditioning set in order to d-separate A from F is the vertex G.

Using the concept of d-separation, one can try to separate a vertex from every other node in a graph G. The smallest set which d-separates an arbitrary vertex V from every other vertex in graph G is called the Markov blanket of V (written as $MB_G(V)$). Since for any node V, the set $MB_G(V)$ d-separates V from the rest of the vertices, it is also the case that statistically speaking the random variables V and U (in the distribution represented by the BN) are independent given the variable set $MB_G(V)$. The Markov blanket of a node V is defined as the set of parents, children, and spouses of V. For example, in figure 2.4, the Markov blanket of the vertex B is the set {A,E,D,C}. Given this, we can conclude that $B \perp G | \{A, E, D, C\}$ and that $B \perp F | \{A, E, D, C\}$.

The markov blanked independencies of a DAG are those independencies that can be generated using the concept of the Markov blanket. Formally they consist of the set $X \perp\!\!\!\perp Y | MB_G(X) : Y \notin MB_G(X), X, Y \in G$. The markov blanket independencies will play an important role in the design of the hybrid algorithm (as discussed in section 3.3.1)

Chapter 3

I-map Hybrid method

This chapter will introduce the main idea behind the hybrid algorithm. As discussed in chapter one, algorithms that perform structure learning fall into two broad categories, each with its advantages and disadvantages. The I-map algorithm tries to take advantage of the benefits of both learning paradigms by combining a score based learner with hard structural constraints.

The I-map algorithm was given its name due to the fact that one of the goals of the algorithm is to construct a BN whose structure is an I-map. Suppose that a BN has a structure G; then G is an I-map of a distribution P if $(X \not\perp\!\!\!\perp Y|S)_P$ implies that $(X \not\perp Y|S)_G$.[20] In other words, when a dependency holds in the distribution, it should also hold in the learned model. Once this goal is achieved, the resulting structure could be examined as is by users in order to find interesting correlations (and possibly design experiments to test some possible dependencies). Figure 3.1 Illustrates the notion of an I-map. When considering different types of BN structures it is important to consider different types of associations. Sometimes an association between variables A and B may be weak or indirect (perhaps A influences C which in turn influences B). A direct association (in a BN) is one in which a variable A is connected to B (so A directly influences B). These associations form the basis of many structure learning algorithms. For example, some constraint based learners work by eliminating direct associations (i.e. by adding constraints that restrict certain variables to be directly influenced by others).

3.1 Statistical dependencies: constraining the search

The I-map algorithm must be able to test different types of dependence hypotheses in the distribution. As with many other learning algorithms, the input given to the algorithm is a data table which is assumed to accurately model the distribution. Note that the data may contain only a small number of rows, in which case this assumption may not hold. In this case, it is still desirable to construct a model which attempts to reconstruct as many features as possible.

Assuming that the data given accurately models the (in)dependence relationships of the distribution, constraints can be gathered from data using a statistical independence test. There are statistical tests available both for discrete and continuous valued data, thus the I-map algorithm can work on learning either type of BN. These tests usually take a relatively large amount of time to compute, so care must be taken in order to ensure that the resulting algorithm may work on relatively large data sets.

Since the goal of the algorithm is to learn an I-map, a dependency fact such as $(X \not\perp\!\!\!\perp Y|S)_P$ immediately poses a constraint on the structure, which in this case implies d-connection of X and Y: $(X \not\perp Y|S)_G$. Given enough of these constraints, a DAG may be constructed guided by a score based search strategy in which no dependency relationships are missed.

3.1.1 Accuracy of the Statistical Test

Statistical tests are used by the I-map algorithm in order to extract dependency information from the data. They are used for the most part as a black box plug-in which can be queried when needed. There are many tests available for performing statistical testing. In the experimental section, the χ^2 test was used for discrete data and the F-test for continuous data. The accuracy of these tests depends largely on the amount of available data. Since this factor is fixed, the statistical tests are programmed to not draw conclusions if the sample size falls below a set threshold [1]. This will drive the type II error rate upwards since many dependencies will not be captured by the statistical test.

It is crucial that the Type I error rate (false discovery rate, or FDR) be as low as possible for the I-map algorithm to work properly. Type II errors on the other hand, can grow considerably large without affecting the performance of the algorithm. Intuitively, if

[1]The threshold is dynamically set for each variable. More detailed information on the thresholds is available in Chapter 4 and 5 where the differences between discrete and continuous models are explained

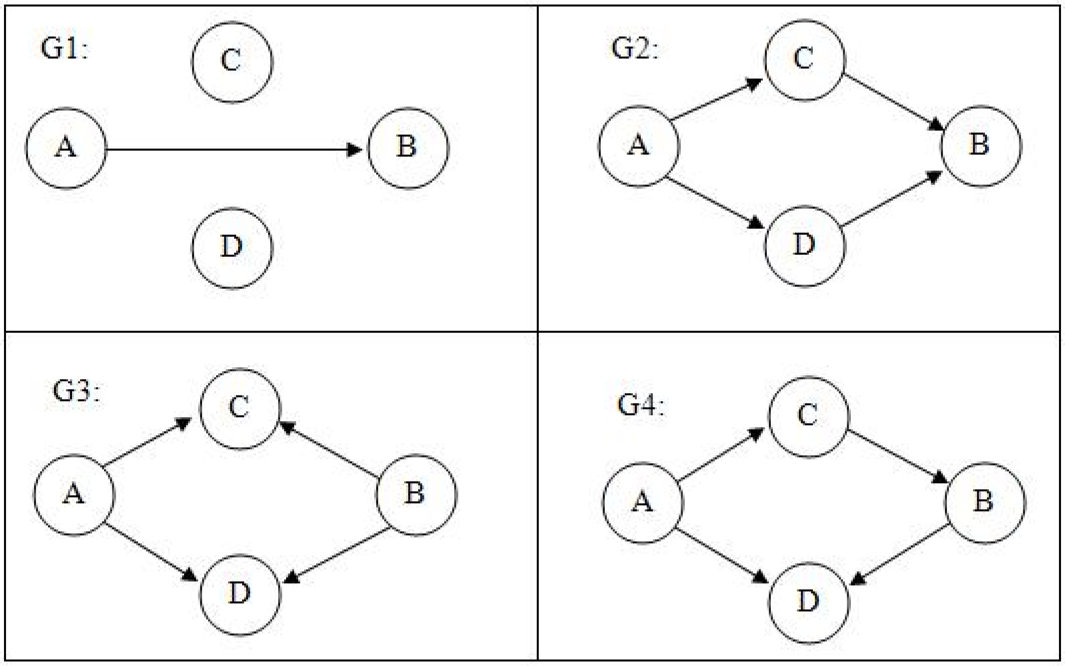


Figure 3.1: Four different configurations of a DAG on vertex set {A,B,C,D}. Assuming that the following dependencies are observed in the data: $\{ A \not\perp\!\!\!\perp B|\{C,D\}, A \not\perp\!\!\!\perp B|\{\}\}$ then the only DAG which is an I-map is G4. Graphs G1, and G2 do not entail $A \not\perp\!\!\!\perp B|\{C,D\}$, while graph G3 does not entail $A \not\perp\!\!\!\perp B|\{\}$

many dependencies are not captured by the statistical test, there is a chance that:

1. They will still be captured by the model via the score function.
2. They will be captured by the model via another (stronger) dependency found by the statistical test.

In practice, a model with fewer parameters (and in this case fewer edges) which is not an I-map is often desirable over one which is an I-map, but has an excessive number of edges, thus a high type II error is not a big concern.

On the other hand, a high FDR will lead to models which are quite poor. This is due to the fact that detecting dependencies not present in the data will lead the hybrid algorithm to include constraints which do not need to be enforced. If many of these false constraints are introduced, the resulting model will have far too many unwanted edges.

Since the number of hypotheses tested can be large, one must take into account the issue of multiple hypotheses testing. Experimentally, conservative schemes like the Bonferroni correction were found to be too conservative yielding a good FDR, at the cost of inferring very few constraints. Other more sophisticated methods such as the ones described in [3] [5] could be implemented. In practice, many constraint based and hybrid methods [18] [6] simply fix the significance level and carry out all hypotheses at this level.

3.2 Search strategy

The search strategy used in I-map is the state of the art GES search strategy [19] [7]. GES is composed of two phases: a growth phase and a shrink phase. Both phases follow a greedy hill climbing style of search. During the growth phase, an initial input structure is given which in many cases is the empty graph (the empty graph is simply a collection of vertices with no edges between them). A score function is used in order to search for local neighbourhoods that strictly increase the number of edges (thus the graph grows in size). GES iteratively chooses neighbours that increase the value of the score function. Once a local optimum is reached, the growth phase terminates and the shrink phase commences. The shrink phase is similar in nature to the growth phase. Both use the same score function to determine which neighbourhood to choose next. The only difference is that in the shrink phase, the graphs considered strictly decrease in number of edges. In the standard GES algorithm there is one growth phase followed by one shrink phase.

Since there are two types of neighbourhoods (one for growing graphs, and one for shrinking graphs) the output at the end of the shrink phase may not be at a local optimum. It may be the case that performing one extra growth phase returns a graph with an improved score function. In theory it is possible to iterate this process several times. For example, one could run the standard GES strategy, followed by another run of GES using the output of the first run as the input to the second run. This process could then itself be iterated any number of times (possibly until a fixedpoint is reached, or until a fixed number of iterations have elapsed). Chickering showed empirically that just one iteration of the growth phase followed by the shrink phase is enough to get good results, and thus the standard GES only has one growth phase and one shrink phase.

3.2.1 The score function

GES is a general search strategy which can accept any type of scoring function. Typically, a score function attempts to fit the learned model as closely to the observed data as possible. In most cases, score functions also carry a penalty term to avoid a model which overfits this data. Score functions for learning BN are usually decomposable [7]. This means that the score S can be decomposed into a sum of terms, each of which is a function of a node and its parents (in the BN structure). For example, a score function S(G) could be written as:

$$S(G) = \sum_{\forall v \in G} s(v, parents(v)).$$

Typically a score function is designed following the assumption that the observed data D is a collection of iid points which sample the target distribution. Morover, the data should also contain the (in)dependencies entailed by the learned model. This is one place where the hybrid approach can aid in learning. Sometimes, the penalty term of the score acts too conservatively and prevents the learned model from adding certain edges, and fitting observed dependencies. By using dependency constraints, the hybrid method ensures that strong observed dependencies are fit by the model.

For the experimental setup (described in chapters 4 and 5), there were two chosen score functions: the BIC and BDeu score functions (both of which are decomposable) [7] [15].

3.2.2 Neighbourhoods and the Search Space

As mentioned before, GES is a greedy local hill climbing strategy for finding locally optimal structures based on a score function. Any generic hill climbing algorithm needs to have a neighbourhood relation in order for the search space to be well defined. Since GES has two phases to its search, two neighbourhood relations are defined.

During the growth phase of GES, a DAG G' is in the neighbourhood of G (notation $nbhd_{+}(G)$) if one can obtain G' from G by a sequence of covered edge reversals, followed by an edge addition, followed by another sequence of covered edge reversals (the concept of edge reversal is explained below) [7]. Similarly a neighbourhood relationship is defined for the shrink phase. During this phase, a DAG G' is in the neighbourhood of G (notation $nbhd_{-}(G)$) if G' can be obtained by edge reversals and the deletion of a single edge.

The neighbourhood relation defines a constrained search space for finding locally optimal solutions. Though this relation is not straightforward (due to the complication of edge

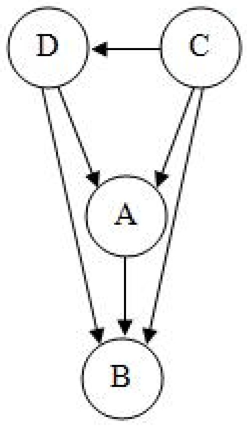


Figure 3.2: Example of a DAG in which $A \rightarrow B$ is covered, but $C \rightarrow D$ is not.

reversals), what one should take away from this is that during the growth phase the number of edges in a DAG increase (strictly one edge at a time) and during the shrink phase this number decreases. Once a local optimum is reached by the shrink phase, the DAG is returned and a parameter estimation algorithm is invoked to complete the BN.

Edge reversals

The operation of an edge reversal is needed because in many cases two distinct DAGs may define the same independence constraints, but one DAG may be a closer fit to the data than the other. An edge $X \leftarrow Y$ is covered if Y has the same parents of X (excluding X itself) (see figure 3.2).

A covered edge reversal is an operation which reverses the direction of a covered edge. In the GES search strategy, covered edge reversal operation are performed on edges in such a way that the resulting DAG is equivalent to the original DAG (in the sense that both DAGs entail the same dependence constraints).

For example, the following DAG structures entail the dependency fact $(X \not\perp\!\!\!\perp Y|\{\})_D$:

1. $X \leftarrow Z \rightarrow Y$
2. $X \leftarrow Z \leftarrow Y$
3. $X \rightarrow Z \rightarrow Y$

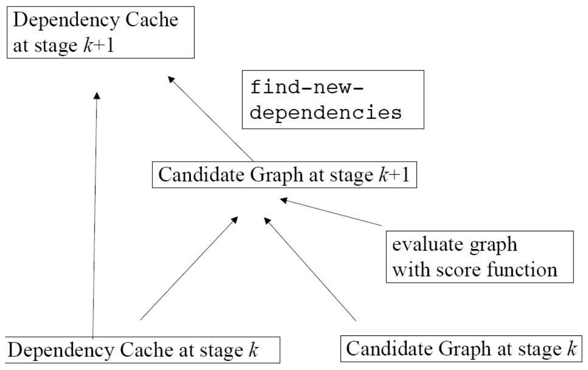

Dependency Cache at stage k+1
find-new-dependencies
Candidate Graph at stage k+1
evaluate graph with score function
Dependency Cache at stage k
Candidate Graph at stage k

Figure 3.3: Diagram of the general I-map algorithm

The first structure suggests that Z may be a common cause (or direct association) of both X and Y, while the second and third structure suggest X and Y are indirectly associated. The edge reversal operation leads to a structure that has the same number of edges as the original and has the same dependency constraints. However, reversing such edges may yield a structure which is closer to the distribution of the data and thus may increase the score function.

3.3 Algorithm design

The I-map algorithm uses GES with a score function to guide the search. Additionally, at each iteration of the hill climbing procedure, dependency constraints are enforced in order to guarantee that the resulting DAG entails every dependence fact found in the data. The algorithm starts by building a base of dependence facts, and at each iteration testing additional hypothesis. A cache of dependency facts is maintained throughout the algorithm so that a particular hypothesis is tested at most once. For example, if at some point the algorithm finds that $(X \not\perp\!\!\!\perp Y|\{U,W\})_D$, this fact is stored into a cache C. At the end of the algorithm the resulting DAG is guaranteed to entail every dependency relation stored in the cache (in this case, the DAG would have X and Y not d-separated given U and W). Figure 3.3 illustrates the algorithm.

The new local optimum of the hybrid search procedure is one in which the score function is maximized given that every dependency fact in the constraint set is entailed by the model. If either condition is not satisfied (that is, the DAG is not an I-map, or the score function can be locally optimized), the procedure must keep searching until a local optimum is reached.

3.3.1 Markov Blanket Dependencies

Theoretically speaking, the algorithm could compute every dependence fact and attempt to construct an I-map out of the entire set of dependencies. However, this is infeasible as the number of conditional dependence tests would grow exponentially large proportional to the number of variables tested. Also, given that performing a statistical test is not trivial, it is desirable to reduce the number of tests needed to construct an I-map. The algorithm dynamically generates and tests hypotheses in order to try to reduce the number of tests performed.

Recall from chapter 2 that in a DAG, a vertex is d-separated from every other vertex when conditioned on its Markov blanket. If a statistical test rejects the hypotheses that $X \perp\!\!\!\perp Y | MB_G(X)$ for arbitrary nodes X and Y in G, then there is strong evidence [2] that the resulting DAG is not an I-map as at least one dependency found in the data is not entailed by the BN. This is the core idea behind the function that generates test hypotheses. If a particular DAG entails all of its markov blanket dependencies, then one can be reasonably certain that an I-map has been found.

As mentioned in the previous section, the algorithm starts with a predefined set of base hypothesis tests. This initial set of hypotheses is fixed and is computed as follows: for every X in the DAG G, and for every Y in G (not equal to X) the test $(X \not\perp\!\!\!\perp Y | \{\})_D$ is performed. In other words, for every variable pair we test whether they are unconditionally dependent. If it is the case that X is dependent on Y, this dependency will be added into the constraint set. This initial set of hypotheses corresponds to the Markov blanket dependencies of the empty DAG. If no hypotheses are rejected (that is, no dependencies have been found) then the empty DAG is indeed a suitable candidate for an I-map. However, it is very likely that some dependencies are found, and thus the search strategy would look for suitable candidates.

Recall that the general GES strategy defines a neighbourhood in which the number of edges in neighbouring graphs differ only by one. This means that at each step of the hill climbing procedure either one edge is added or one removed (depending if we are in the grow or shrink phase), and it also means that the markov blanket of any arbitrary node does not change too much. Once a candidate DAG is chosen by the search algorithm, new

[2] Statistical testing is not 100% accurate, thus one can only say that there is strong evidence supporting a fact.

hypotheses are generated by testing the markov blanket dependencies of the two nodes X and Y adjacent to the newly added (deleted) edge. In addition, hypotheses are generated for the nodes adjacent to either X or Y. These are the only nodes whose markov blanket has potentially changed. The find-new-dependencies procedure implements these rules in order to generate new hypotheses. Once the hypotheses are generated, a statistical test is invoked, and those hypotheses which are rejected are added into the constraint set.

3.3.2 Enforcing constraints

In order to guarantee that the hybrid algorithm returns a DAG which is an I-map, the dependencies stored in the constraint set must be enforced throughout the course of the algorithm. The general way in which this is achieved is by modifying the way in which the search algorithm chooses the next DAG in the neighbourhood.

During the growth phase, GES attempts to modify the DAG (and improve the score) by adding edges. Since this is the only operation that affects dependencies[3], it is clear that the number of dependencies (or constraints) entailed by the new neighbour do not decrease. Since a DAG which maximizes the score may not cover all the dependencies in the constraint set, it is necessary to add a statement which ensures that the growth phase for the hybrid algorithm does not stop when a local optimum is reached. In other words, the search strategy must be able to pick neighbours which do not necessarily improve the score function, but which cover strictly more dependencies than the original DAG. In this sense, the constraints are enforced by modifying the neighbourhood relationship, and by relaxing the score criterion.

At the end of the growth phase, the algorithm returns a candidate I-map (all constraints are satisfied). However, this model is likely to be an overfitted version of the true model. If the shrink phase of the standard GES algorithm was used, there is a risk that some of the dependencies covered by the overfitted model may be lost. This is due to the fact that the goal of the score function is to maximize the fit of the model to the data while minimizing the number of parameters. This does not guarantee that all dependencies will be observed in the model. In order to ensure that the result of the shrink phase is an I-map, the neighbourhood relationship is modified so that only DAGs which cover every dependency

[3]The forward steps may also change the direction of the edges. In this case, the edge reversal operations are done conservatively, so they do not affect the number of dependencies covered, and thus these operations can safely be ignored in the analysis

in the constraint set are considered. Once this neighbourhood is fixed, the score function can safely be maximized.

In this thesis, the hybrid method has been implemented on a model selection type of learner (that is, a learner which returns a single BN). However, other algorithms exists (such as model averaging algorithms [20],Ch8) in which a single BN is not returned. Model averaging returns a set of BN's which may be a good fit for the data,together with a posterior likelihood assigned to each model. When performing inference, single BN inference algorithms are applied to each individual model, and the results are multiplied by the different posterior probabilities. Model selection is computationally less costly and under some assumptioms it acheives good results [20],Ch8. One of the strengths of the hybrid algorithm is that it can be applied to (just about) any structure learning algorithm. In the case of model averaging. The constraints would be applied to each model, and thus every conceibable model would be an Imap of the given data.

Chapter 4

Imap Learning on Discrete Bayesian Networks

This chapter discusses the implementation, and experimental set up for discrete BN.

4.1 Algorithm

The hybrid algorithm for discrete BN follows closely the high level description of the general algorithm described in chapter 3 (section 3). It uses the generic GES search strategy together with the well established BDeu and BIC score functions [7] [15]. The procedure find-new-dependencies is in charge of generating statistical hypotheses and testing them. Those hypotheses which are rejected (i.e., the dependencies found) are added to the constraint set C. During the growth phase of the algorithm, a neighbour is chosen if

1. The score function increases, or
2. The number of dependencies covered (by the model) in the constraint set increases.

Whenever there is a choice between a neighbour which increases the score, and one which increases the number of dependencies covered, the former is preferred over the latter (always choosing the neighbour whose score value is highest). The local maximum will eventually be reached, at which point the algorithm checks if every dependency in the constraint set has been covered. If this is the case, the shrink phase can be invoked. Otherwise, the neighbour with highest score value which increases the number of covered dependencies is chosen.

Once a local optimum is reached (every dependency is covered, and no neighbour has a higher score value), the shrink phase is invoked. During this phase neighbours are chosen in a way that dependencies covered in the previous phase are not lost. The pseudocode for the algorithm is defined as follows:

Algorithm 1 The hybrid I-map procedure for discrete BN.

```
Input: data sample d for random variables V.
Calls: score evaluation function Score(G,d), statistical testing procedure find-new-dependencies(G,d).
Output: BN structure that maximizes score function given dependency constraints.
1: Initialize with the Empty graph over V.
2: for all Variables A,B do
3:   test the hypothesis $A \perp\!\!\!\perp B|\{\}$
4:   if $A \perp\!\!\!\perp B|\{\}$ is rejected by statistical test, add dependency into constraint set C
5: end for
6: {growth phase}
7: while local optimum not reached do
8:   if ∃ a neighbour G' whose score is higher than current DAG G then
9:     choose G' with maximum score
10:  else if dependency constraints in C are not covered by G then
11:    choose G' in the neighbourhood (which covers more dependencies in C, but has
       lower score than G) with maximum score
12:  else
13:    local optimum reached, return
14:  end if
15:  $C := C \cup$ find-new-dependencies$(G, d)$
16: end while
17: {Shrink phase}
18: while local optimum not reached do
19:   if ∃ a neighbour G' whose score is higher than current DAG G, which covers all
      constraints in C then
20:     choose G' with maximum score
21:   else
22:     local optimum reached, return
23:   end if
24:   $C := C \cup$ find-new-dependencies$(G, d)$
25: end while
```

4.2 Statistical testing

For discrete BN, the statistical test used to determine dependency relations was the χ^2 test [10]. This test is not reliable when the available amount data is scarce. For this reason, the algorithm only uses dependencies whose reliability is above a certain threshold. The threshold value was set using standard statistical practices [10], Ch 9.1. For the χ^2 test to be reliable, the expected number of samples in each cell must be at least 5; in other words, $m * p_i \geq 5$ (where p_i is the probability of cell C_i according to the null hypotheses and m is the sample size). Given that this fact holds, the number of type I errors can be kept to a minimum. Section 4.4 shows empirically that the FDR is indeed kept at acceptably low rates for various settings.

4.3 Other Methods for Learning Discrete BN

It has been observed that many score based methods tend to underfit the target structure [6] [1] [7]. This may be a direct result of the penalty term associated with the score which aims to keep the complexity down by penalizing structures with large numbers of parameters. Constraint based methods such as PC and CPC also tend to underfit the target structure. This is partly due to the fact that such methods rely on independencies to constrain the search space. Since statistical tests typically tend to accept hypotheses even when there is a reasonable chance that the hypotheses should be rejected [1], the Type II error tends to be large on small to medium sample sizes. Because of this, many edges are removed and the resulting structures tend to be sparse.

4.4 Experimental results

This section describes the empirical evaluation of the hybrid method for discrete BN. The code was written in java, and uses tools from the Tetrad package [23]. The algorithm has been tested with both synthetically generated data as well as with real structures.

[1]Statistical tests typically use a significance level of at most 5%. This means that even with a low probability (say 6%) of observing the actual data given that the null hypotheses is true, this hypotheses will not be rejected

4.4.1 General setup

In order to test the performance of the hybrid method the following setup was used:

1. First, a structure (the target DAG) is either randomly generated, or given as input.
2. A random assignment of parameters is generated and assigned to the structure to construct a BN. The assignment of parameters is chosen from an uniform distribution. For each table, the parameters are then normalized so they follow standard random variable properties (i.e. the probability of the sum of all instances of an event add up to 1, and the values are non-negative).
3. Data is randomly generated following the distribution entailed by the target BN.
4. Several structure learning algorithms attempt to reconstruct the target structure given the available data.
5. Once the structure is learned, a maximum likelihood parameter estimator is used to assign parameters to the learned structure and give a complete BN.
6. The learned BN's are compared to the target BN as well as to their data fit in order to assess the quality of the learned models.

The structure is generated using an upper bound on the number of edges. We have set this upper bound to twice the number of vertices. This follows the experimental set up of [22]. The idea is to have a graph which is neither too sparse nor too dense (in order to simulate realistic structures). The degree of the edges is allowed to take on any value (as long as the maximum number of edges is not exceeded). Unless stated otherwise, the discrete experiments are performed on binary valued BN's. In order to determine how well the learners perform, the edges (adjacencies) from the learned structure is compared against the adjacencies from the target structure. There are two types of error which can be made by the algorithm. Either there are edges in the learned structure which are not present in the target BN (false positive), or there are edges in the target structure which do not appear in the learned BN (false negative). The edges which appear in both the learned and in the target structure are called true positives. To help interpret the results, these numbers are combined into one composite value through the use of the F-measure [27] p.164. The

F-measure is defined as:

$$\frac{2 * TruePositive}{2 * TruePositive + FalsePositive + FalseNegative}$$

Another measure used throughout the experiments is the Kullback-Leiber (KL) divergence of the model to the true distribution [17]. This measure has been used in previous studies [6] [1] to estimate how close the model is to the true distribution. In order to use the KLD measure, data is randomly generated using the distribution entailed by the learned BN; then, this data is compared to the input dataset (using the standard KLD measure). The KLD measure is widely used in evaluating the performance of BN's, such as in [6] and [26]

4.4.2 Results

Score based methods underfit

The first experiment illustrates the issue of underfitting with score based learners mentioned in section 4.3. In this case the target DAG is a simple collider of the form $X \rightarrow Z \leftarrow Y$. The I-map algorithm uses the BDeu score, the score based learner employs the GES search strategy together with BDeu score function. Both algorithms use default parameters following [7] for the score function. The random variables in the DAG were assumed to be ternary (that is they take on 3 values). Datasets of different sizes were generated, ranging from 100 to 2000 points. For each fixed sample size, different data was generated with different parameter settings (there were a total of 1000 different datasets per sample size). The percentage of graphs with correct adjacencies (that is graphs which have exactly two edges X–Z and Y–Z, without taking into account the direction) was calculated. The result is plotted on figure 4.1. The I-map algorithm performs better than the score based counterpart when small to medium datasets are given (sizes less than 1800). After this point, the difference in performance is very small. The improvement in performance is greatest in the low data sizes (100 to 500 data points), with improvements ranging between 20% to 30%. This simple example illustrates that the BDeu method often fails to fit statistically significant dependencies. By making sure that such dependencies are covered, the Imap algorithm obtains a structure which is closer to the target model than the structure learned by the BDeu learner.

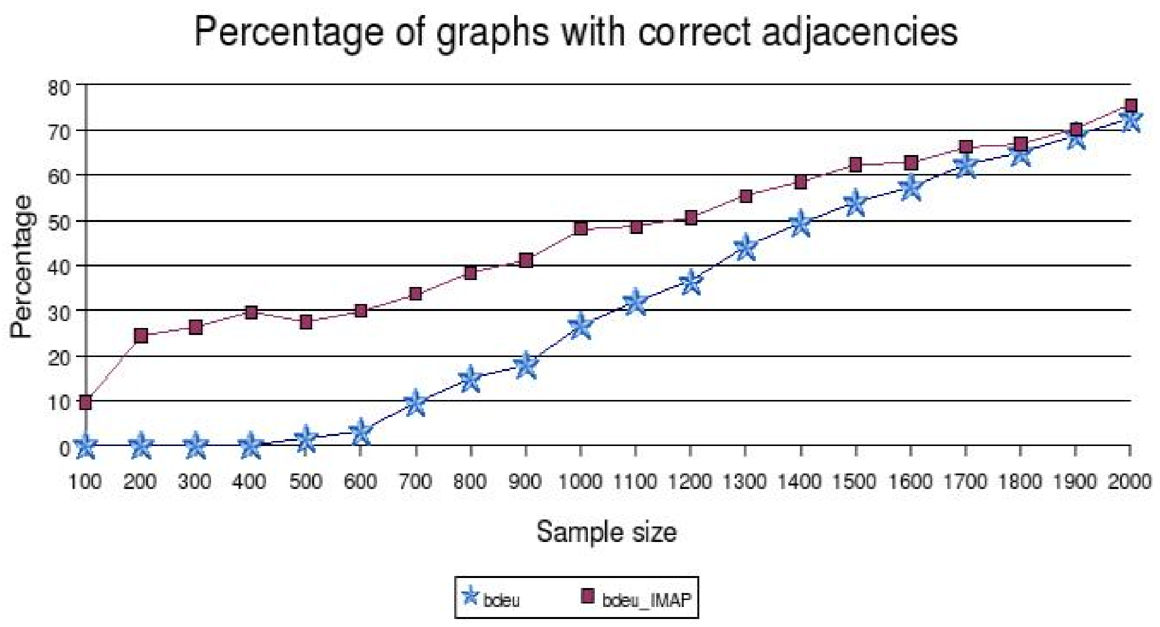


Figure 4.1: The graph illustrates how score based methods can underfit even simple structures due to the penalty term. The hybrid algorithm consistently performs better when the sample size is less than 1800 points.

Synthetically Generated Structures

The next experiment shows that the hybrid method obtains structures that are consistently closer to the target structure than those learned by competitor methods. The experiment setup follows the general setup described in section 4.4.1. Randomly generated DAGs were generated as target structures to be learned. The size of these graphs ranges from 4 to 10 nodes inclusive and the size of the randomly generated data ranges from 100 to 8000 points. For each fixed sample size, and for each fixed graph size 30 different DAGs were generated and relearned by the algorithms. The statistical test used in the Imap algorithm was χ^2 with a significance level of 5%. The average performance was calculated and is summarized in the figures that follow.

Figure 4.2 shows the average improvement in edge F-measure for various settings. There is a pronounced difference in the lower sample sizes, which gets smaller as the sample size increases. The graph in the bottom right suggests that the improvement in F-measure is independent of the size of the graph (since the trends are nearly horizontal). The graph in the top right shows how the improvement is greatest (with a gain between 0.3 to 0.4) in the

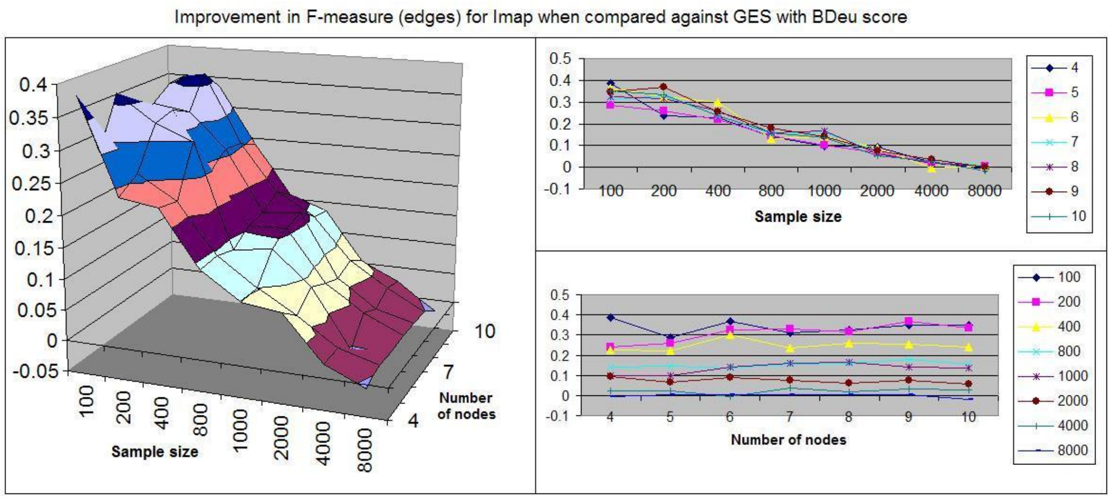


Figure 4.2: This figure shows the average improvement in F-measure (for edges) of Imap over the GES algorithm (both using BDeu score) plotted against sample size and number of nodes. The values are in the range [-1,1] with positive numbers representing the magnitude of improvement achieved by Imap, 0 meaning no improvement, and negative numbers meaning that the hybrid method performs worst than GES. The figure in the left shows a 3-D view of the results of the experiment. On the right, two 2-D trends are plotted, one using sample sizes for the values of the x-axis and one with number of nodes.

lower sample sizes. The performance of both methods seem to converge when the sample size reaches 8000 data points.

Figure 4.3 shows in detail the effect of sample size on the improvement in F-measure. All the values were averaged and grouped by sample size. The trend clearly shows that there is a pronounced difference in lower to middle sample sizes.

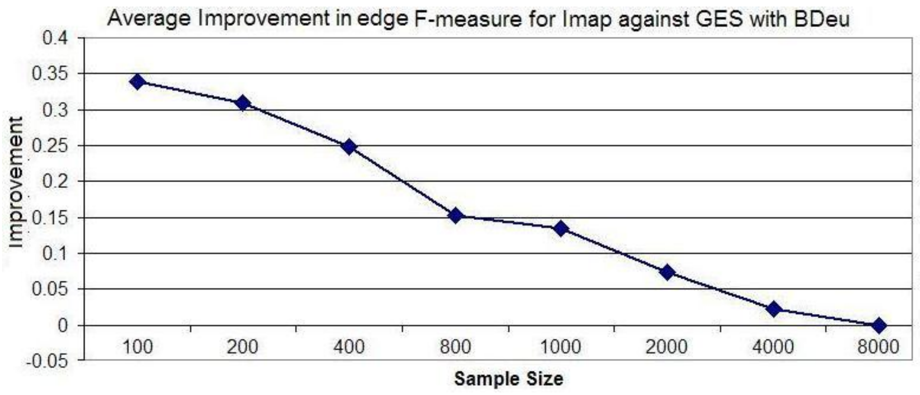


Figure 4.3: Average improvement in edge F-measure of Imap over the GES algorithm (both using BDeu score) plotted against sample size.

The next figure (Fig 4.4) shows the difference in KLD between GES (with BDeu) and Imap (BDeu).

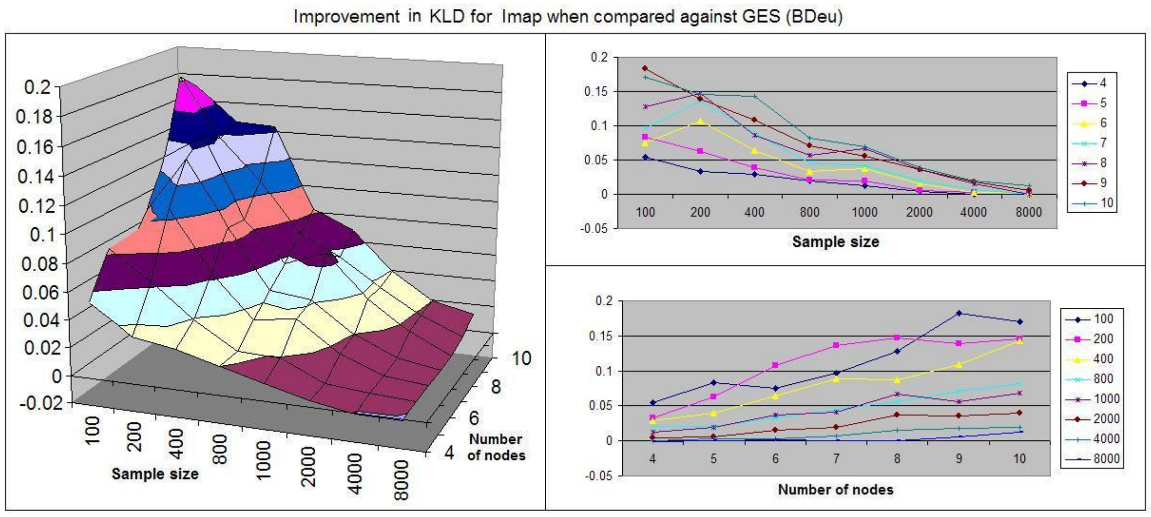


Figure 4.4: This figure shows the average improvement in KLD (for edges) of Imap over the GES algorithm (both using BDeu score) plotted against sample size and number of nodes.

Similarly to figure 4.2 higher values indicate that the Imap algorithm is a closer fit to the distribution, while negative values suggest otherwise. In order to assess how close the learned graphs are to the target distributions, an independent set of data was randomly generated using the target graph (as the distribution to follow). Both graphs were compared to this independent dataset, and their fit to the data was recorded.

The graphs suggest that both sample size and number of nodes in the DAG play a role on the magnitude of the improvement. Similar to the F-measure, the greatest improvement is achieved in the lower sample sizes. In contrast to the F-measure, the magnitude of the improvement seems to grow with the size of the graphs.

Since the Imap algorithm is a generic algorithm which can be adapted to any score function (and any search strategy), it was also tested with the BIC score. The figures below are identical to those presented above. Imap (with BIC score) is compared against the GES algorithm (also with BIC score). The average improvement in edge F-measure, and in KLD were recorded and plotted.

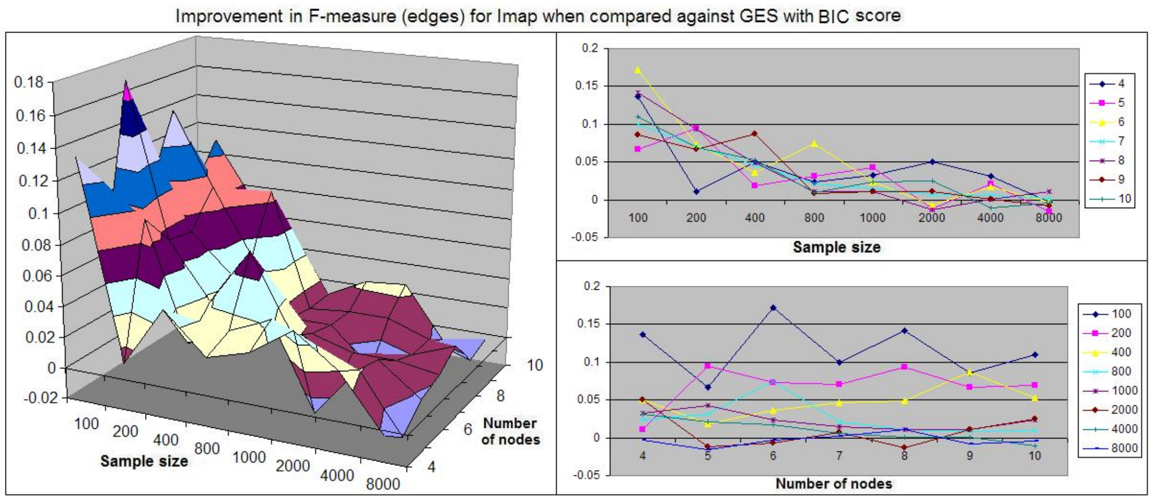


Figure 4.5: This figure shows the average improvement in F-measure (for edges) of Imap over the GES algorithm (both using BIC score) plotted against sample size and number of nodes. The values are in the range [-1,1] with positive numbers representing the magnitude of improvement achieved by Imap, 0 meaning no improvement, and negative numbers meaning that the hybrid method performs worst than GES. The figure in the left shows a 3-D view of the results of the experiment. On the right, two 2-D trends are plotted, one using sample sizes for the values of the x-axis and one with number of nodes.

The improvement in F-measure is not as large as that achieved by the hybrid algorithm on the BDeu score. It is still a significant improvement (more than 10%) especially in the lower sample sizes. There does not seem to be a strong correlation between graph size and the magnitude of improvement. However, the improvement is correlated to the sample size, having the greatest magnitude in the lower to middle sample sizes, and converging to zero when the number of data points reaches 8000. Figure 4.6 illustrates this phenomenon. The trend displays the average improvement (over all graph sizes) against sample size.

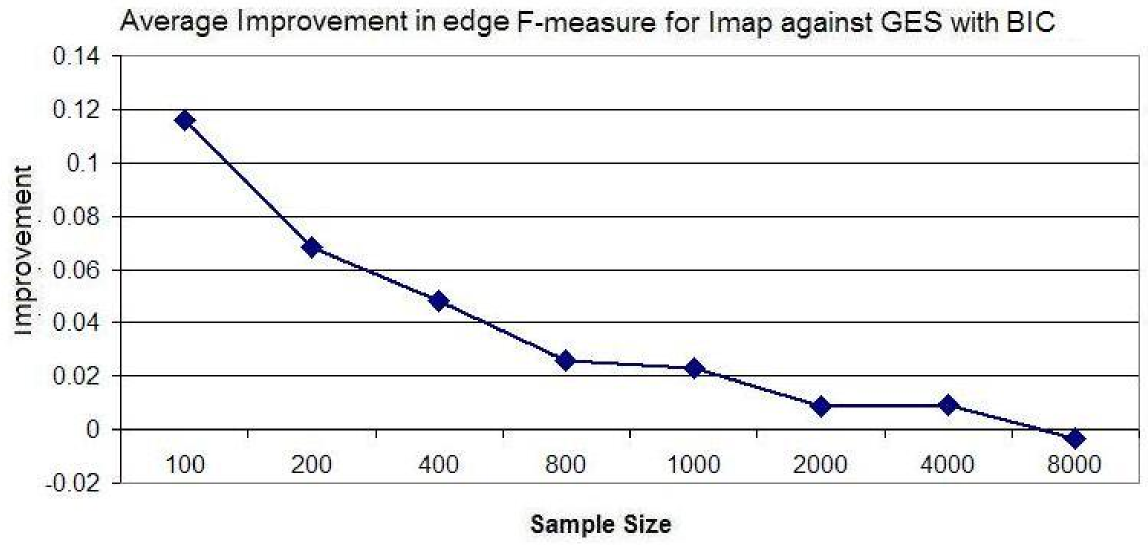


Figure 4.6: This figure shows the average improvement in F-measure (for edges) of Imap over the GES algorithm (both using BIC score) plotted against sample size.

The graphs in figure 4.7 show the improvement in KLD of Imap when compared against GES (with BIC score). The trend is similar to the one seen in when the BDeu score (figure 4.4) but with a smaller magnitude.

The improvement is greatest on small sample sizes, with the larger graphs. As the sample size increases, there is little improvement (values are very close to zero). There is a fairly large area in which little improvement is observed (sample size greater than 1000, and number of nodes less than 9). This is due to the fact that with these settings, the BIC score produces a model which is very close to the target distribution. Figure 4.8 shows a table with selected KLD values for the GES algorithm. The sample sizes shown range from 1000 to 8000, and the graph sizes range from 4 to 8 nodes. Most values get very close to zero indicating that the GES algorithm (with BIC score) does indeed produce a model which is

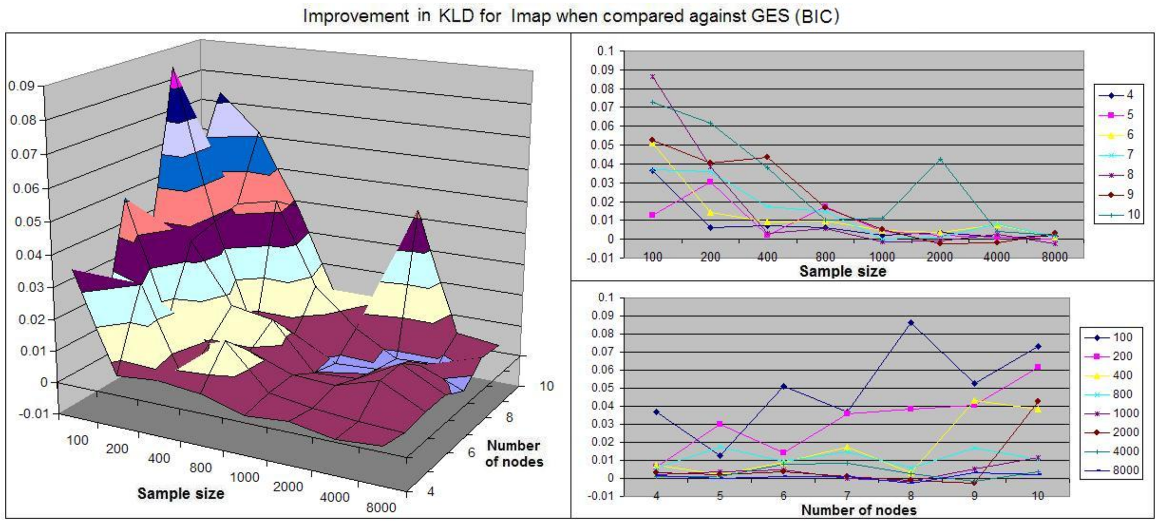
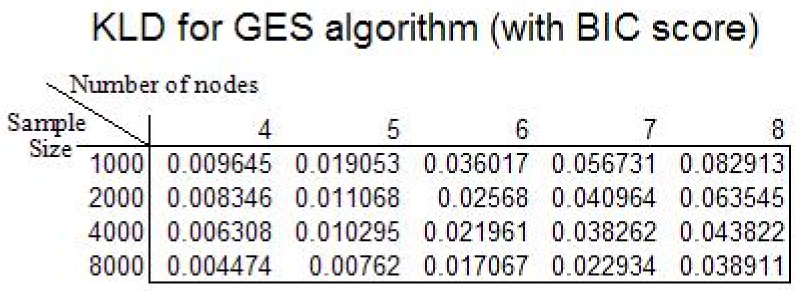


Figure 4.7: This figure shows the average improvement in KLD (for edges) of Imap over the GES algorithm (both using BIC score) plotted against sample size and number of nodes.

very close to the target distribution. When larger graphs are used to generate data (and implicitly to model a distribution), the distribution modeled becomes more complex. This implies that in order to capture the correct distribution, more data is needed, and that the models learned may not fit the distribution as well as when small DAGs are used.


KLD for GES algorithm (with BIC score)

Sample Size \ Number of nodes	4	5	6	7	8
1000	0.009645	0.019053	0.036017	0.056731	0.082913
2000	0.008346	0.011068	0.02568	0.040964	0.063545
4000	0.006308	0.010295	0.021961	0.038262	0.043822
8000	0.004474	0.00762	0.017067	0.022934	0.038911

Figure 4.8: Table of values of KLD for the GES algorithm.

As seen in the previous experiments, the Imap algorithm does seem to improve the quality of the learned models. Intuitively, this is due to the fact that some dependency missed by the competing algorithm was captured by Imap. In order to quantify the strength of the missed dependencies, the mutual conditional information $I(X, Y|Z)$ was computed. The average value is plotted against sample size and number of nodes in figure 4.9.

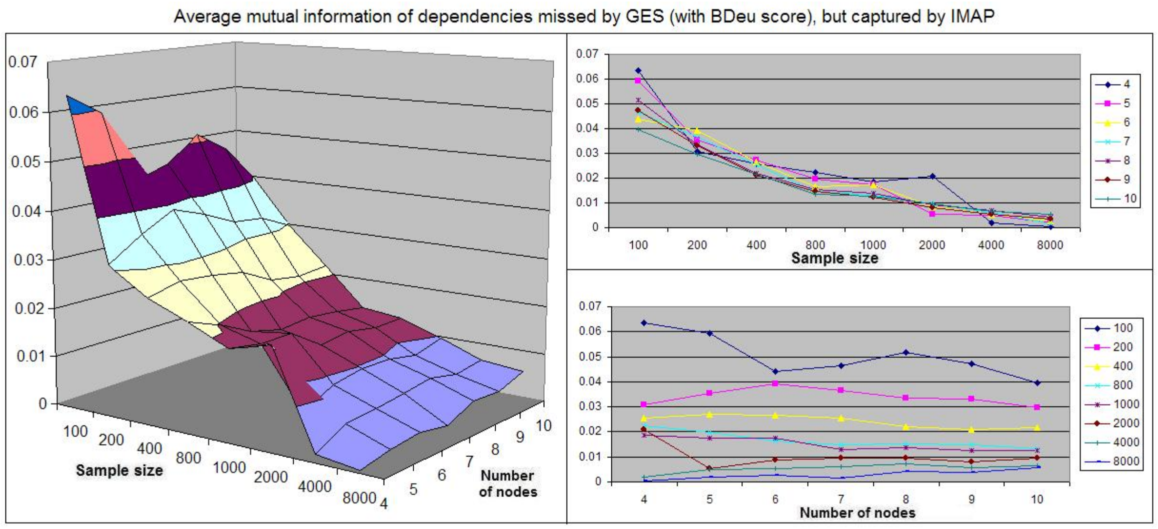


Figure 4.9: Average mutual information plotted by sample size and number of nodes.

The trend in the top right shows that as more data is available for learning, less dependencies are missed by the GES algorithm. In the smaller sample sizes, it can be seen that some important dependencies are indeed missed by GES (at sample size 100, in 4 node DAGs the average mutual information was greater than 0.06). The size of the graph seems to have no effect on the strength of missed dependencies. The fact that GES misses some strong dependencies in the lower sample sizes helps explain why the Imap algorithm helps to improve the structure and distribution fit of the learned models. For medium sample size range, the strong dependencies have been captured by both methods; however, there are still a number of weak dependencies that have not been captured by the GES algorithm. These dependencies are likely to be indirect constraints posed on the structure (for example, and edge needed to complete a collider between two variables) as opposed to direct constraints.

Performance of the Statistical Test

While running experiment on synthetically generated data, we also captured information on the performance of the statistical test. As mentioned in section 3.1.1 the performance of the statistical test depends largely on the performance of the statistical test (more precisely on the FDR). Below, figure 4.10 shows the average FDR for a particular test run.

The FDR seems to remain fairly stable as graphs grow large (the trend shows no signs of increasing). The trend on the top right suggests that there is a tendency towards higher FDR

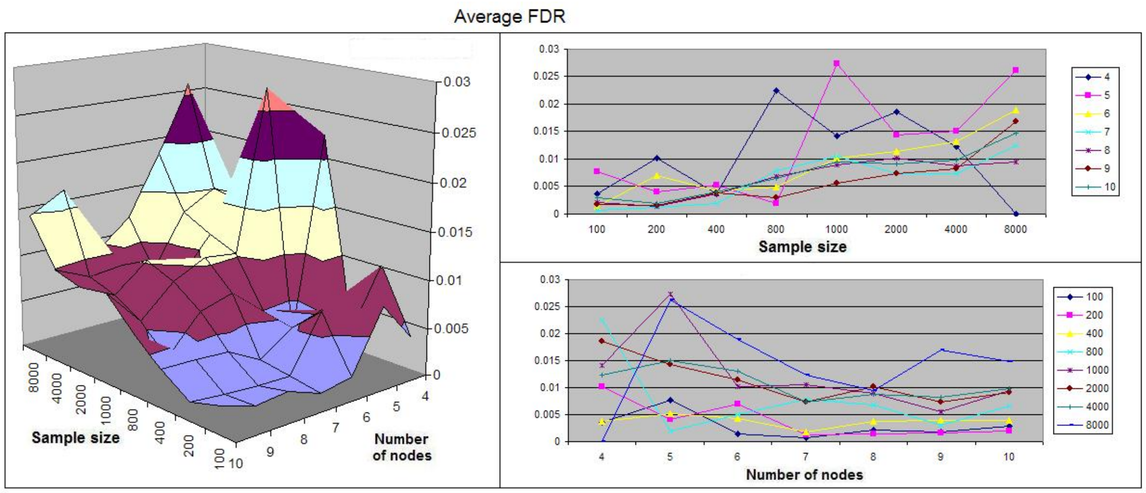


Figure 4.10: Plots showing the average FDR. Right: two 2-D figures show the average trend grouped both by number of nodes and sample size. Left: the 3-D view of the graphs on the right.

as the sample size becomes larger. This may be the result of the statistical test becoming oversensitive to correlations, and could potentially be controlled by methods which control the error on multiple hypotheses testing (such as the methods described in [3] or [5]). In our experiments the FDR always remained below the set value for the significance level (5%), thus no such methods were needed.

Simulations with Real World Networks

This section describes the experiments made with two well known real world BNs: Alarm [2] and Insurance [4]. The testing setup was the same as the one with the synthetically generated networks. The performance was measured using both F-measure and KLD. The figures presented in the next pages summarize the performance of the hybrid method when compared against GES (with BDeu score). These are large graphs having more than 20 nodes per graph (insurance has 25 vertices, and alarm has 37).

These figures give a brief snapshot of the results for real world networks. Though the improvement of the hybrid method is small, the DAGs returned by it are I-maps of the distribution. We expect that by implementing a dynamic strategy for controlling the FDR, the Imap algorithm would get a larger improvement.

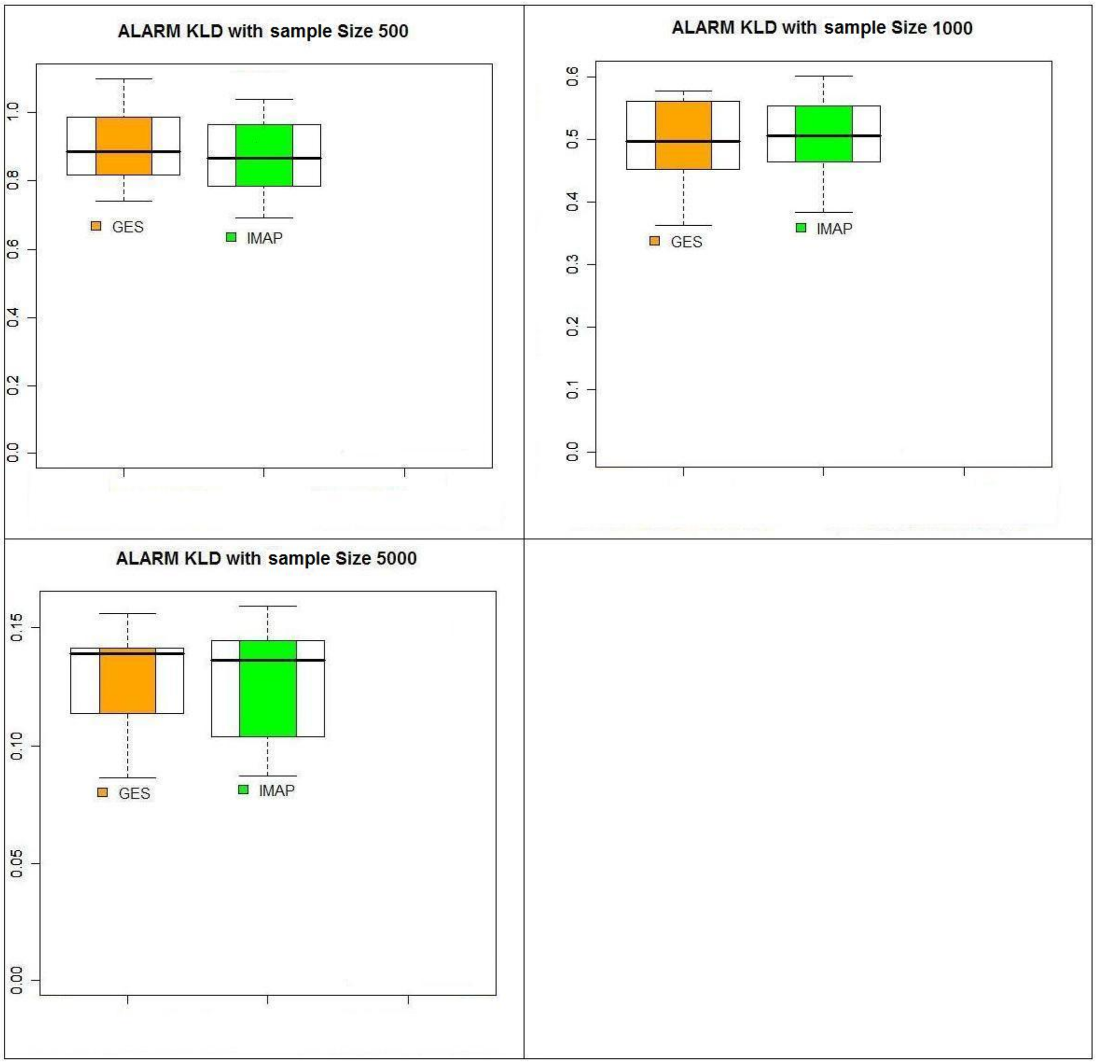


Figure 4.11: Boxplots comparing the KLD measure in the alarm network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. The measures are very close to each other, but Imap obtains a slightly better KLD in the lower setting (sample size = 500). Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

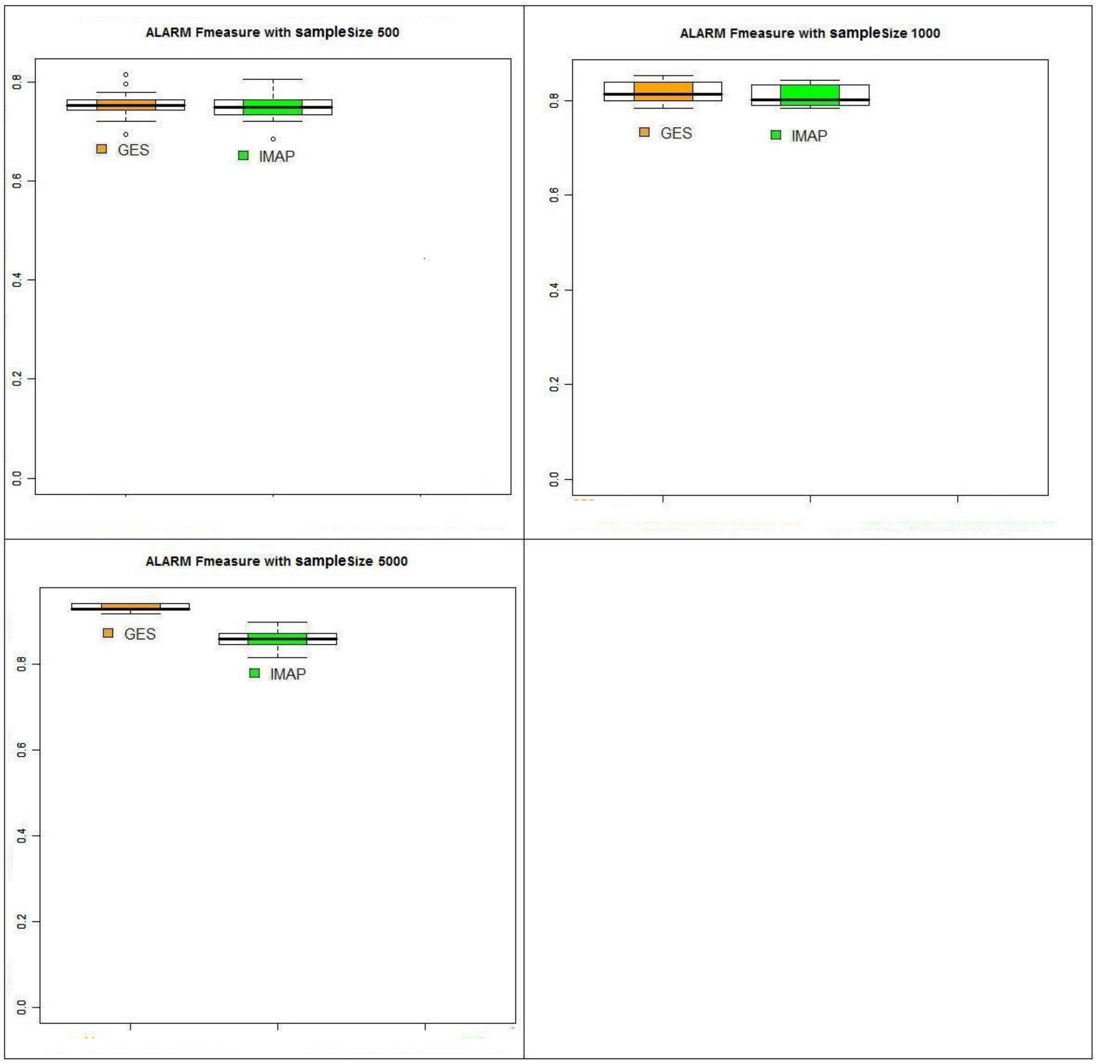


Figure 4.12: Boxplots comparing the F-measure in the alarm network for 3 different sample sizes. Higher values indicate a better F-measure. In this case the measures are nearly identical in all settings (with the exception of the 5000 sample size setting. In this particular case, the hybrid algorithm may be underfitting the structure which in turn results in a lower fMeasure.)

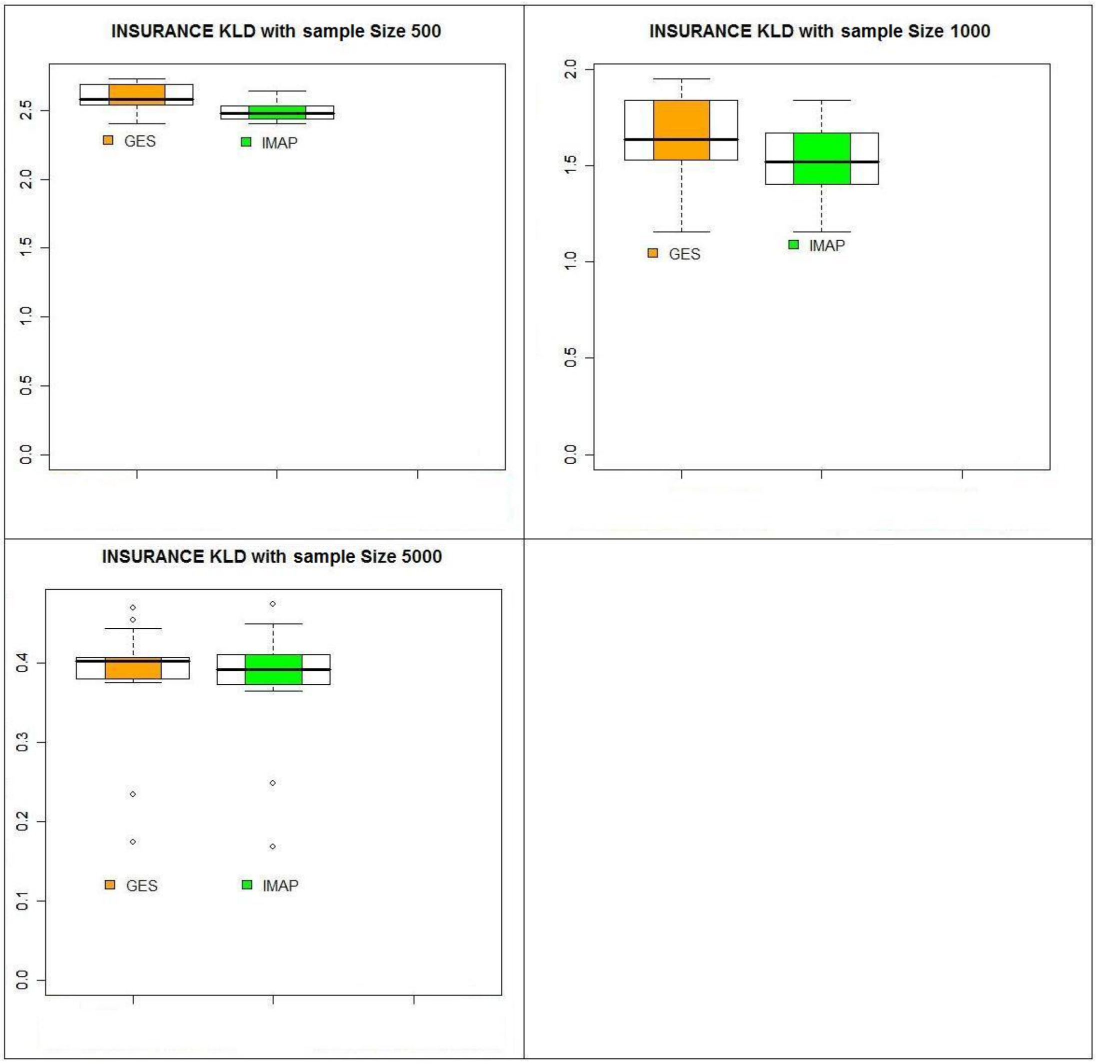


Figure 4.13: Boxplots comparing the KLD measure in the insurance network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. The measures are very close to each other, but Imap obtains better KLD in all settings. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

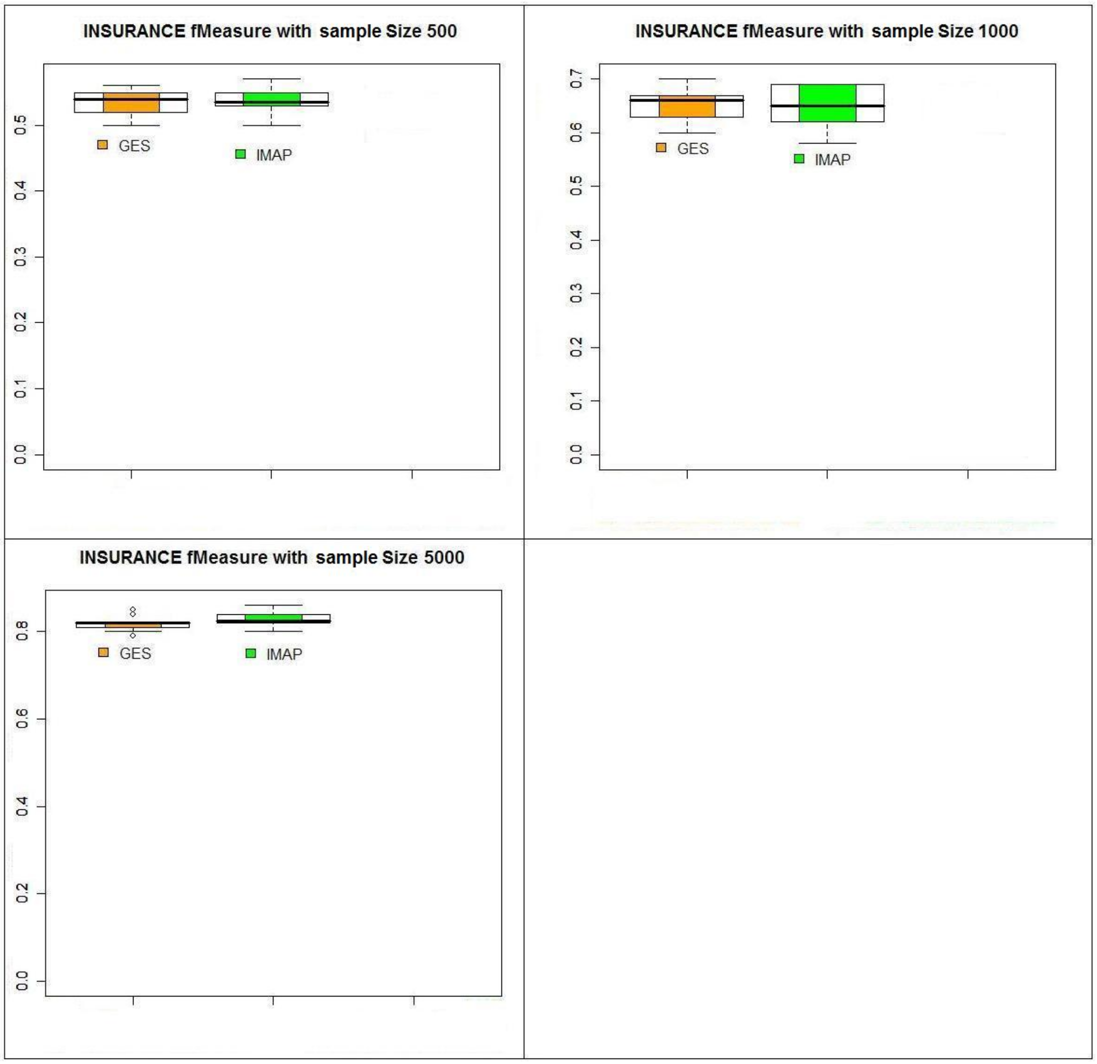


Figure 4.14: Boxplots comparing the F-measure in the insurance network for 3 different sample sizes. Higher values indicate a better F-measure. In this case the measures are nearly identical in all settings (at sample size = 5000, the F-measure for Imap is just above that of the F-measure of GES). Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

Chapter 5

Imap Learning on Continuous Bayesian Networks

This chapter discusses the implementation, and experimental set up for continuous BN (SEM).

5.1 Algorithm

The hybrid algorithm for SEMs follows closely the high level description of the general algorithm described in chapter 3 (section 3). It uses the generic GES search strategy together with the well established BIC score function. (This is the only standard bayesian score, thus it is the only function used in the testing phase). Similarly to discrete BN, the procedure find-new-dependencies generates statistical hypotheses and tests them. Those hypotheses which are rejected are added to the constraint set C.

The growth phase of the continuous version of the Imap algorithm is slightly different than the growth phase of the discrete version. During the growth phase of the algorithm, a neighbour is chosen only if the number of dependencies covered (by the model) in the constraint set increases. Among the neighbours which increase the number of dependencies covered, the one with the highest score value is chosen. The reason for this is that unlike the score methods for learning discrete BNs, score based learners tend to produce models which overfit the given data.

Once a local optimum is reached (every dependency is covered), the shrink phase is invoked. During this phase neighbours are chosen in a way that dependencies covered in the previous phase are not lost. In addition to the shrink phase defined by GES, a prune phase is performed following the general idea of the SIN approach [12]. Section 5.1.1 provides more detail of the pruning phase. Two versions of this algorithm have been tested. One version simply stops after the pruning phase and returns the structure. The other version takes the DAG returned after the prune phase and performs one additional grow phase followed by an additional shrink phase. The reason for incorporating an additional grow-shrink phase is to guarantee that the resulting DAG does fits every dependency entailed in the constraint set. The pseudocode for the algorithm is defined as follows:

Algorithm 2 The hybrid I-map procedure for continuous BN.

```
Input: data sample d for random variables V.
Calls: score evaluation function Score(G,d), statistical testing procedure find-new-dependencies(G,d).
Output: BN structure that maximizes score function given dependency constraints.
1: Initialize with the Empty graph over V.
2: for all Variables A,B do
3:   test the hypothesis $A \perp\!\!\!\perp B|\{\}$
4:   if $A \perp\!\!\!\perp B|\{\}$ is rejected by statistical test, add dependency into constraint set C
5: end for
6: {growth phase}
7: while G does not cover all dependencies in C do
8:   choose G' in the neighborhood with maximum score
9:   $C := C \cup$ find-new-dependencies$(G, d)$
10: end while
11: {Shrink phase}
12: while local optimum not reached do
13:   choose G' in neighborhood with maximum score
14:   $C := C \cup$ find-new-dependencies$(G, d)$
15: end while
16: {Prune phase}
17: for all Variables A,B do
18:   if $A \perp\!\!\!\perp B|\{\}$ or $A \perp\!\!\!\perp B|\{V - X, Y\}$ then
19:     remove edge A–B (if any) from G
20:   end if
21: end for
22: {optional}
23: perform another growth phase (like the one above)
24: perform another shrink phase (like the one above)
```

5.1.1 Pruning

During the prune phase, the tests $X \perp\!\!\!\perp Y | \{\}$ and $X \perp\!\!\!\perp Y | \{V - \{X, Y\}\}$ are performed. If either hypothesis is rejected, this is reasonable evidence that there should be no edge between variables X and Y since having X and Y independent to each other implies that in a SEM, an edge between X and Y is incorrect.

Note that the implication only goes one way. Assuming hypothetically that the statistical independence test made no mistake, if an independency fact says that X and Y are independent, then a BN which can generate the distribution should not have a link between X and Y. However, if a SEM does not contain a link between X and Y, this does not imply that X and Y are independent. Figure 5.1 provides an example illustrating this point.

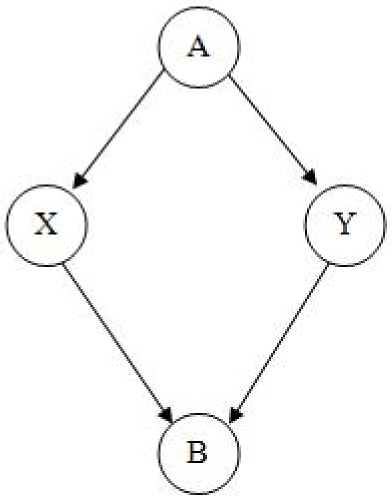


Figure 5.1: In this DAG, there is no link between X and Y; however, it is easy to see that $X \not\perp\!\!\!\perp Y | \{\}$ and $X \not\perp\!\!\!\perp Y | \{A, B\}$. The only independency between X and Y is observed when conditioning on A; that is, $X \perp\!\!\!\perp Y | \{A\}$

Even when testing several independence hypotheses, the fact that none of them yield an independence relationship between X and Y does not guarantee that an edge should be present. This is one of the main reasons why the method described in [12] is used only as a pruning mechanism.

The main idea of the pruning mechanism is to lower the overfitting factor introduced by the score metric. However, this may remove some dependencies fitted in previous stages and therefore two options are available at this point:

1. Either the DAG is returned as is with no theoretical guarantees, but with results that suggest that the edges observed in the DAG are of very high quality
2. an additional grow-shrink phase is performed in order to guarantee that the returned DAG is an I-map.

The result for both algorithms are examined in the experimental section of the chapter. For the remainder of the thesis, the version of Imap which returns a model after pruning will be called Imap-pruned, and the other simply Imap.

5.2 Statistical testing

As with the discrete case, the hybrid algorithm requires the use of a statistical test. In this case, we followed other constraint based methods and employed Fisher?s z-statistic for testing whether a given partial correlation is 0 [25] Ch 5.5. This test has the property of being more reliable in small sample sizes [13] CH 5. The following graphs plot both the FDR (Type I error rate) and FAR (Type II error rate).

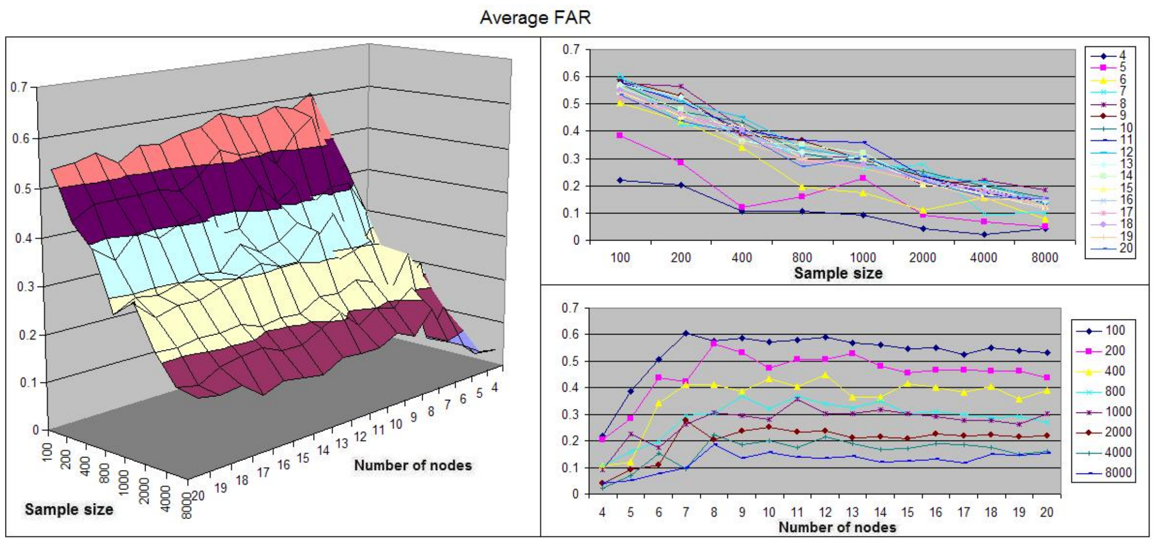


Figure 5.2: The average FAR. The trends suggest that the size of a DAG (in terms of number of nodes) is correlated with the FAR only on the smaller DAGs. For graphs with 8 or more nodes, the FAR remains constant (and independent of node size). As expected the FAR is correlated with sample size having larger rates on smaller sample sizes which grow small as the sample size increases.

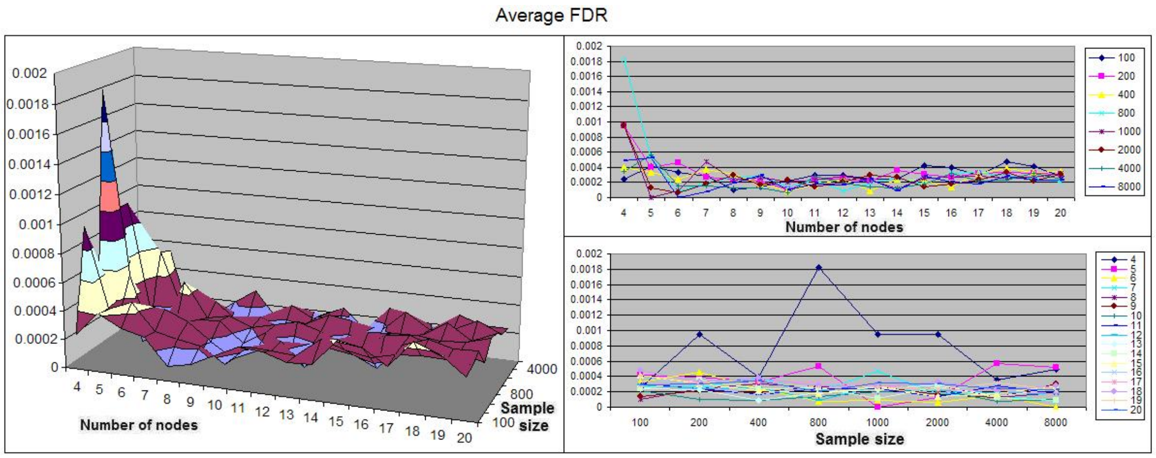


Figure 5.3: Average FDR. The figure suggests that the FDR is well behaved and within acceptable rates. The FDR is largest on smaller graphs (5 or less nodes) but quickly decreases and stabilizes itself

The graphs suggest that the Type I error rate is whitin acceptable ranges, thus the dependencies found by the cache tend to be reliable. On the other hand high type II error rates can have a negative effect on the Imap-pruned algorithm especially on smaller graph sizes with small sample sizes (where the FAR is the highest). Indeed in the experimental section for the imap-pruned algorithm (section 5.3.1) it becomes evident that some underfitting occurs at these levels. On the other hand, the pure Imap algorithm corrects the underfitting introduced by large type II errors by having an extra growth phase. Given that the FDR is very low, we can see that the resulting model covers a large range of true dependencies.

5.3 Experimental Results

5.3.1 General Setup

The experimental setup follows closely the set up of the experiments for discrete BN (see section 4.4). A DAG is either randomly generated or given as input. Parameters are uniformely assigned, with a range of [-1,1] in order to obtain an SEM. Data is randomly generated (following the distribution entailed by the SEM) and given as input to the various learning algorithms. The goal is for the learning algorithms to match the generating structure from the given data set. Most of the experiments are larger (that is, they cover larger graphs sizes) than the ones performed in the discrete case experimental setup. The reason for this is that the hybrid algorithm seems to outperform GES on larger graph sizes. In addition, the statistical test used in the continuous case is much faster that the one used for learning discrete BN, which makes the entire testing procedure much faster then in the discrete case.

Score Based methods overfit

This section illustrates the main problem with score based methods: overfitting. In order to measure by how much a model overfits, the number of edges is compared to the number of edges in the target model. Figure 5.4 shows the ratio of the number of edges returned by GES over the number of edges in the target DAG.

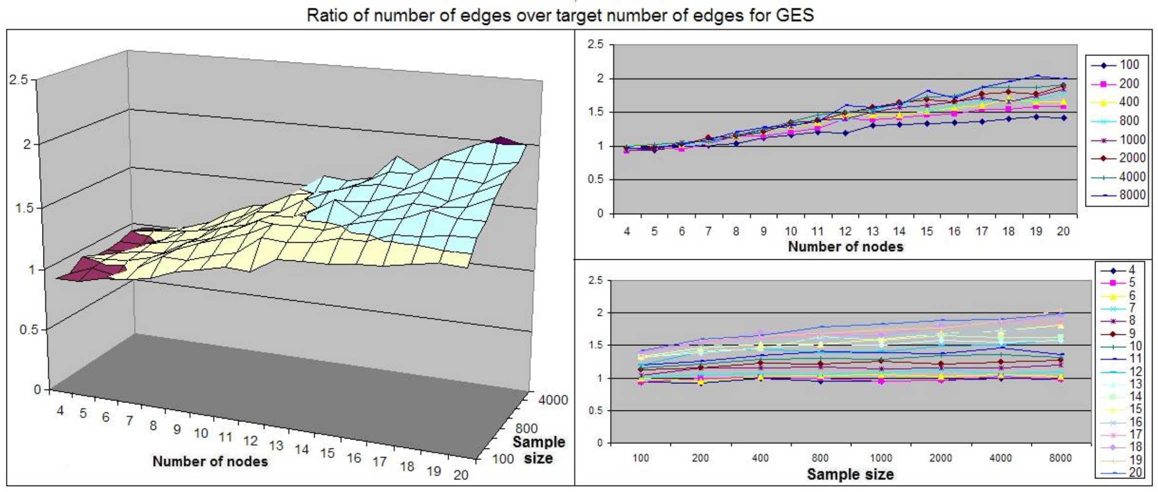


Figure 5.4: The overfitting factor of the GES algorithm. The figure shows the number of edges returned by GES over the average number of edges in the target DAG. A ratio of 1 is ideal indicating that GES has the same number of edges as the target DAG. Values below 1 suggest underfitting, and values over 1 indicate overfitting. The trend in the upper right shows a clear tendency that as the DAG becomes larges, so does the overfitting factor. The figures also seem to suggest that for the larger graphs, the overfitting factor grows with the sample size.

In lower sample sizes, and lower graph sizes, the GES algorithm seems to be performing well. However, it is clear from the figure above that as the number of nodes increases the overfitting problem becomes more acute. In fact, at some points the GES algorithm returns a model with more than twice the number of edges as the target model. This trend is summarized in figure 5.5 where the average ratio is computed and plotted against the number of nodes in a DAG.

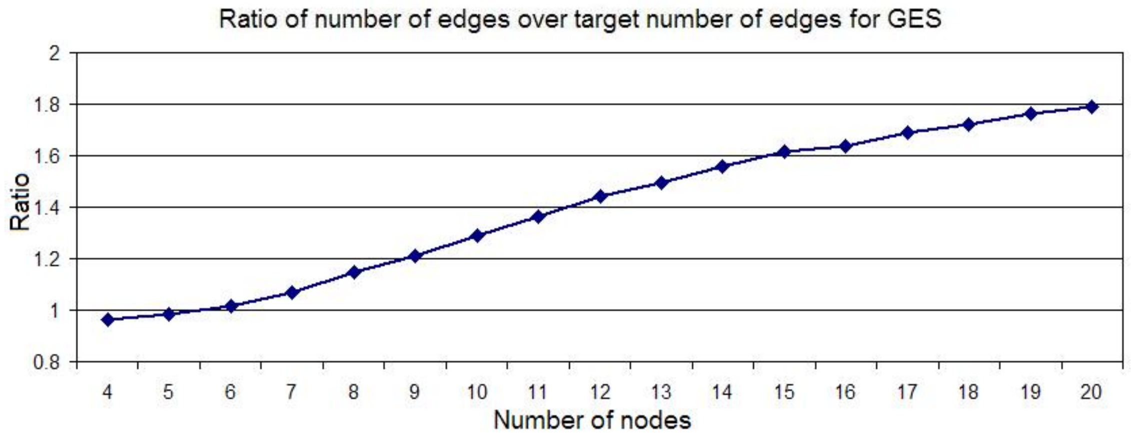


Figure 5.5: The figure shows the average number of edges returned by GES over the number of edges in the target DAG plotted against number of nodes. As the graphs become larger, the overfitting factor becomes worse.

Synthetically Generated Structures with Imap-pruned

The next experiment shows that the hybrid method (Imap-pruned) obtains structures that are closer to the target structure than those learned by competitor methods. The experiment setup follows the general setup described in section 4.4.1. Randomly generated DAGs were generated as target structures to be learned. The size of these graphs ranges from 4 to 20 nodes inclusive and the size of the randomly generated data ranges from 100 to 8000 points. For each fixed sample size, and for each fixed graph size 30 different DAGs were generated and relearned by the algorithms. The statistical test used in the hybrid algorithm was Fisher's z statistic with a significance level of 5%. The average performance was calculated and is summarized in the figures that follow.

Figure 5.6 shows the average improvement in edge F-measure for various settings. The largest improvement occurs in the larger DAGS (9 nodes or larger). There is a small dip in performance in the smaller graphs. This is due to the fact that GES performs very well with graphs with small number of nodes, having average F-measure values above 85% for many of the settings (see figure 5.8 for actual values). As the number of nodes increases, the overfitting factor starts affecting the GES procedure. This is where the Imap-pruned algorithm achieves the best results improving the F-measure by as much as 18%. Figure

5.7 shows in detail the effect of graph size in the improvement in F-measure. All the values were averaged and grouped by graph size. The trend clearly shows that there is a tendency to higher improvement as the graphs get larger in size.

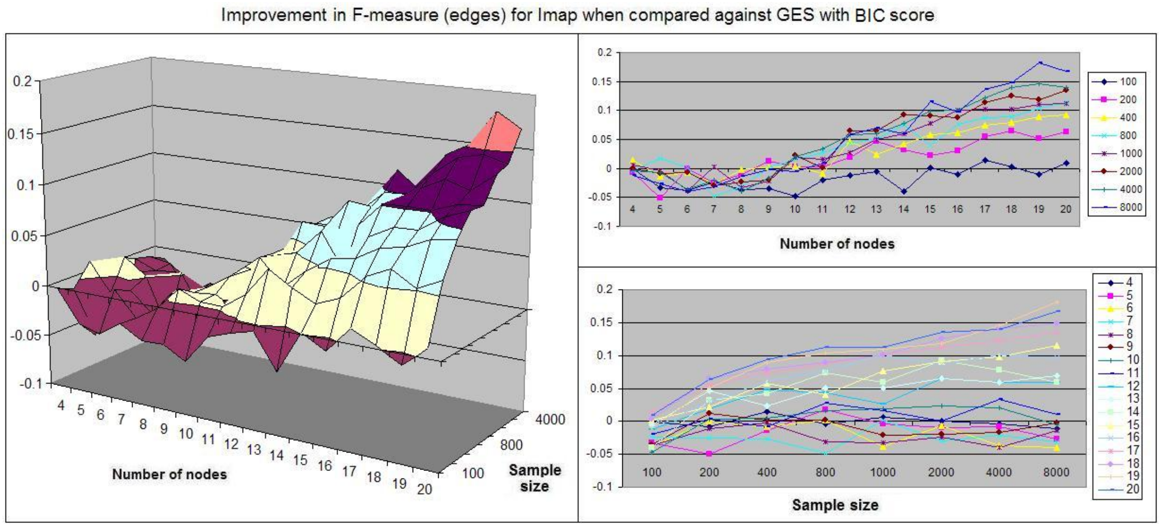


Figure 5.6: This figure shows the average improvement in F-measure (for edges) of Imap-pruned over the GES algorithm (both using BIC score) plotted against sample size and number of nodes. The values are in the range [-1,1] with positive numbers representing the magnitude of improvement achieved by Imap-prunned, 0 meaning no improvement, and negative numbers meaning that the hybrid method performs worst than GES. The figure in the left shows a 3-D view of the results of the experiment. On the right, two 2-D trends are plotted, one using sample sizes for the values of the x-axis and one with number of nodes.

It is clear from the figures above that Imap-pruned does improve the structural measures (at least for the larger graphs). In addition, the models learned by the Imap-pruned algorithm typically have about the right number of edges. Figures 5.9 and 5.10 display the ratio of learned edges over edges in the true graph.

The hybrid algorithm has a strong influence over the underlying GES algorithm and successfully controls the overfitting problem introduced by the scoring method. For the most part, the Imap-pruned algorithm seems to be obtaining about the right number of edges. For very small sample sizes (100 and 200) the algorithm returns models which have fewer edges than the target graph. This may be due to the fact that large type II errors affect the prune phase of the algorithm making it too aggressive (see section 5.2) for more info.

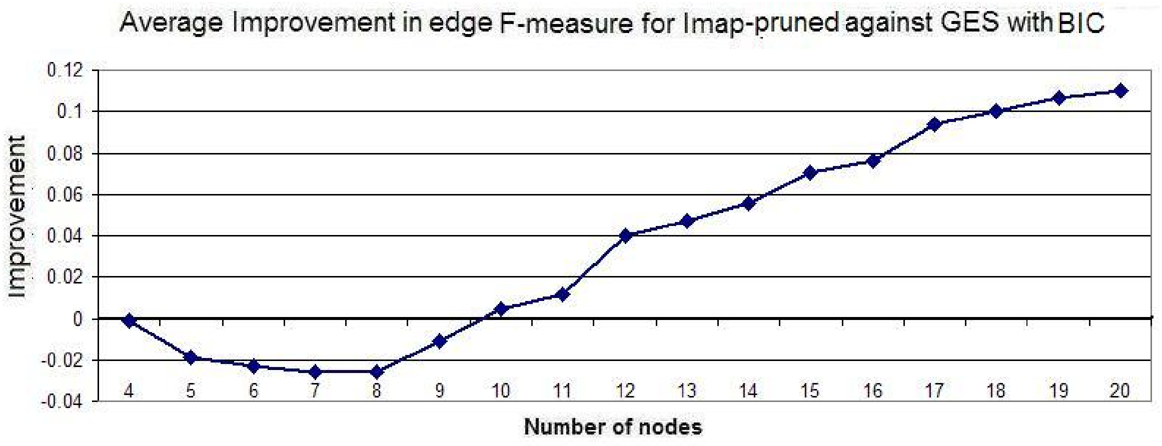
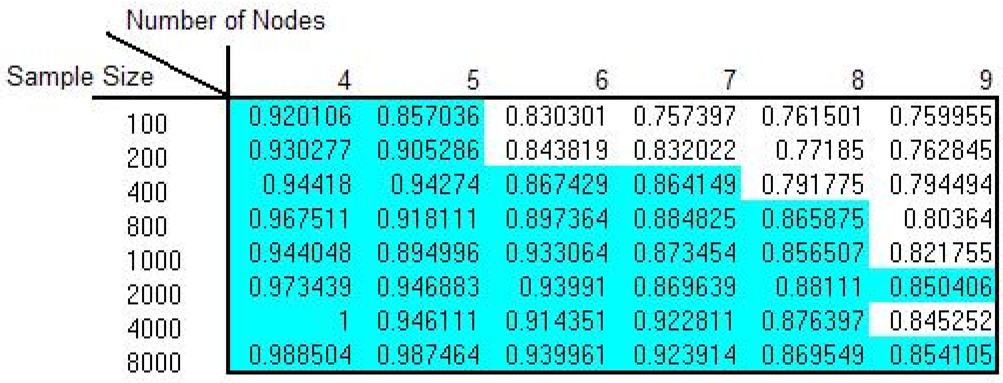


Figure 5.7: Average improvement in edge F-measure of Imap-pruned over the GES algorithm (both using BIC score) plotted against number of nodes.



Sample Size \ Number of Nodes	4	5	6	7	8	9
100	0.920106	0.857036	0.830301	0.757397	0.761501	0.759955
200	0.930277	0.905286	0.843819	0.832022	0.77185	0.762845
400	0.94418	0.94274	0.867429	0.864149	0.791775	0.794494
800	0.967511	0.918111	0.897364	0.884825	0.865875	0.80364
1000	0.944048	0.894996	0.933064	0.873454	0.856507	0.821755
2000	0.973439	0.946883	0.93991	0.869639	0.88111	0.850406
4000	1	0.946111	0.914351	0.922811	0.876397	0.845252
8000	0.988504	0.987464	0.939961	0.923914	0.869549	0.854105

Figure 5.8: Average edge F-measure for GES (with BIC score). The values above 85% have been shaded. This table shows that GES performs well for graphs with relatively few nodes.

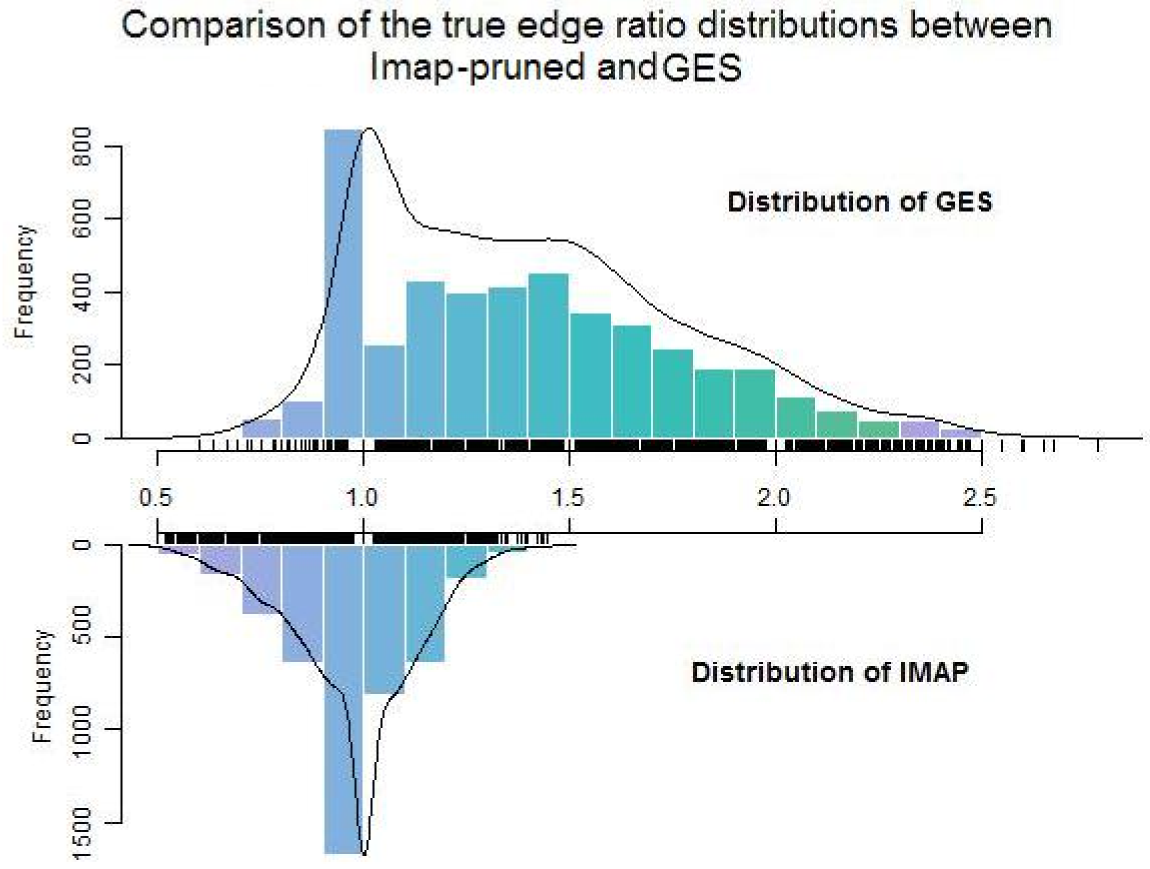


Figure 5.9: Comparative histograms for edge ratios between GES and Imap-pruned. The Imap-pruned method has a strong influence over the distribution. Recall that the ideal case is to have a ratio of 1 (meaning that the learned model has the same number of edges as the target. The Imap-pruned algorithm seems to have a distribution of edge ratios which follows a Normal distribution centered at 1. In contrast, the GES algorithm has 2 modes and has a large skewed to the right (towards larger graphs).

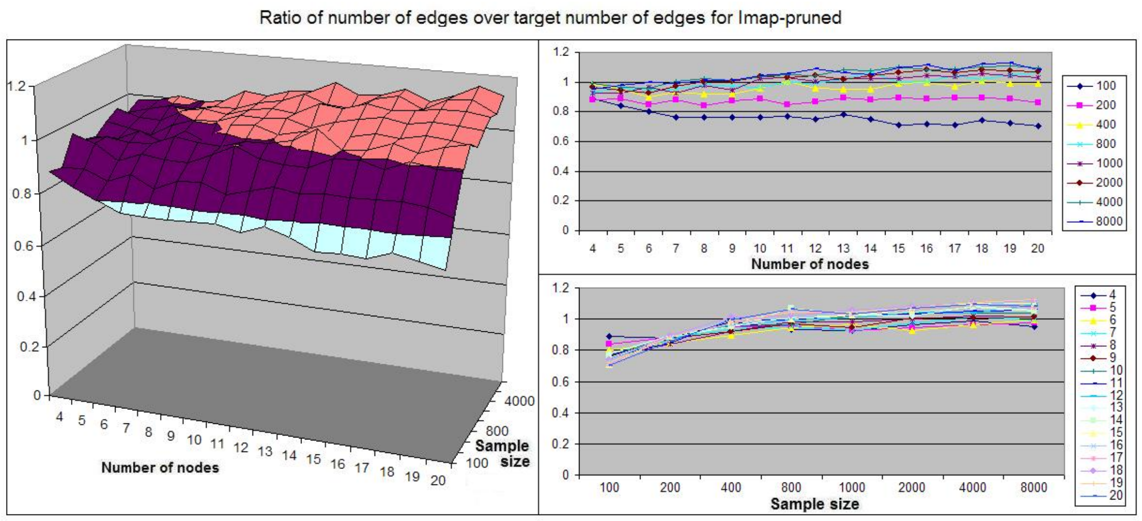


Figure 5.10: The different charts show the average ratio of edges learned over the target number of edges. The trends suggest that the ratio is independent of the graph size. Also, it seems that for very small sample sizes the Imap-pruned algorithm tends to underfit (due to relatively large type II error rates). However, with larger sample sizes, the algorithm seems to be obtaining about the right number of edges.

5.3.2 Synthetically Generated Structures with Imap

Recall that the Imap algorithm differs from the Imap-pruned algorithm in the last phase. Instead of returning a pruned model, Imap performs an extra growth (and shrink) phase in order to guarantee that the resulting model fits all the dependencies collected in the cache. The drawback with this approach is that the structural F-measure is not as strong as the one observed with the pruned version of the algorithm. In fact, there seems to be little improvement in edge F-measure when compared against GES. Figure 5.11 shows boxplots comparing the F-measure distributions for both algorithms.

The Imap algorithm does have advantages over using the regular GES algorithm. For example, the structure returned is much simpler (has fewer edges than that of the one returned by GES). In addition, the divergence measure is very similar to that of GES indicating that it is a close fit to the data but with a simpler structure. Figures 5.12 and 5.13 illustrate this point.

The strongest point of the Imap algorithm remains the fact that it is an Imap. Together with the low FDR rates given by the statistical test, we can see that it is a good Imap. It does not fit the structure as closely as the Imap-pruned method does, but it helps improve

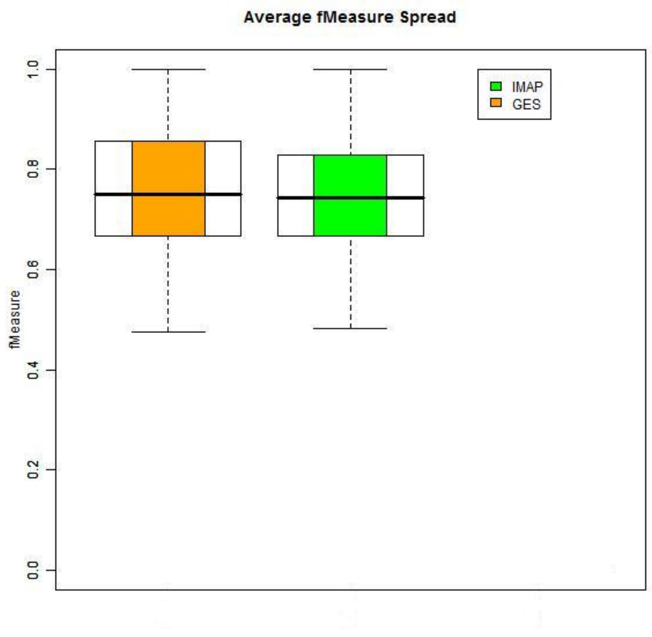


Figure 5.11: Boxplots comparing edge F-measures between GES and Imap.

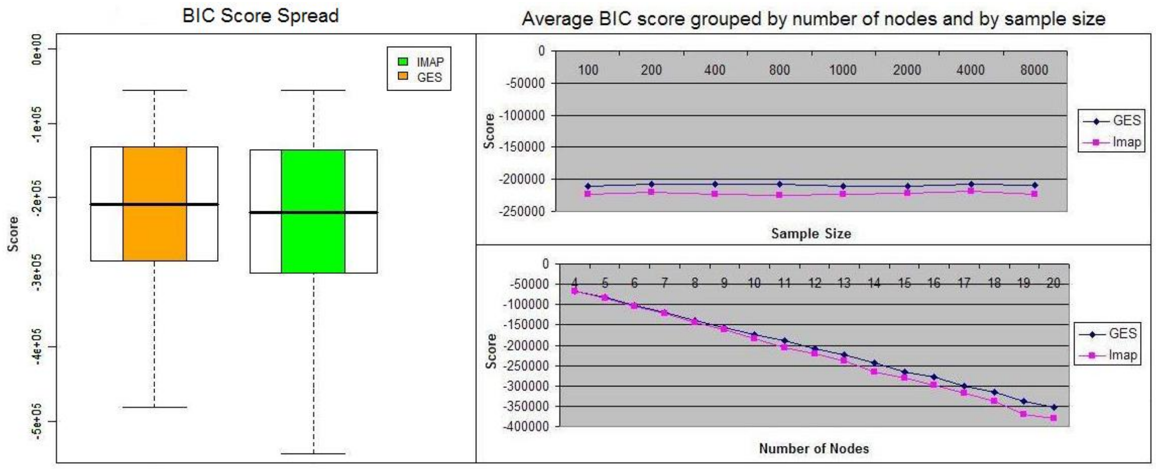


Figure 5.12: Trends comparing the average BIC score between GES and Imap. As can be seen a relatively small penalty (in terms of BIC score) is paid for a simpler structure which covers all the dependencies entailed by the cache.

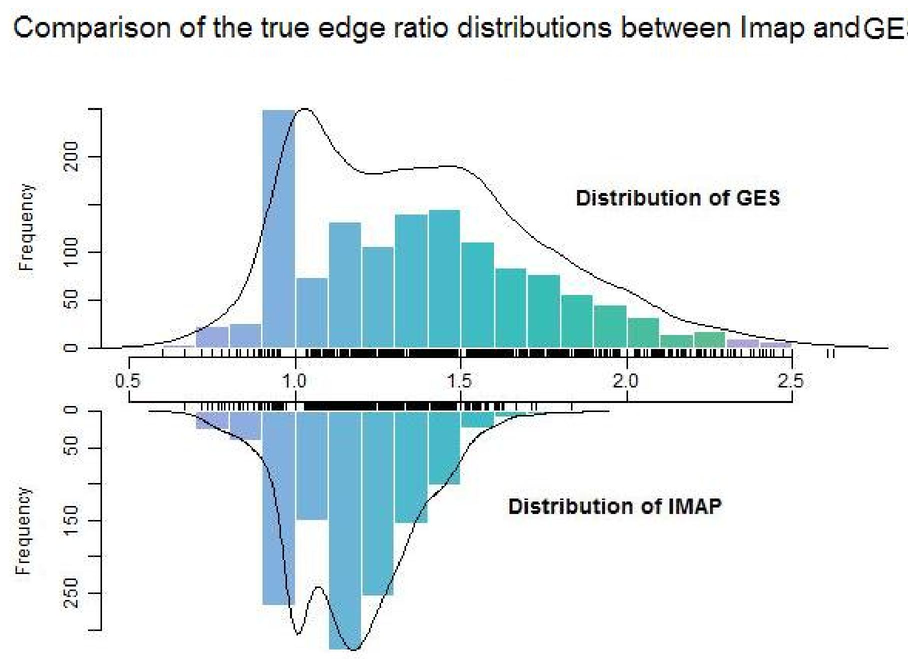


Figure 5.13: Comparison between the distribution of edge ratios between GES and Imap. The Imap algorithm substantially reduces the overfitting factor.

the overfitting problem introduced by the scoring method. One possible explanation for not performing as well as the Imap-pruned method is that some dependencies (which are indeed likely to be true in the distribution) are fitted with an edge which is not part of the target DAG. Further experiments are needed in order to fully understand and solve this problem, but one direction that seems promising is to include some independencies constraints in order to further refine the search procedure (the Imap-pruned method was very successful thanks to these constraints).

5.3.3 Simulations with Real World Networks

As with discrete BNs, this section describes the experiments made with two well known real world BNs: Alarm [2] and Insurance [4]. The testing setup was the same as the one with the synthetically generated networks. The performance was measured using both F-measure for structural analysis, and BIC score as a divergence measure.

Imap-Pruned

The following four figures show how the Imap-pruned algorithm performs. In general, the F-measure is better on all settings (for both alarm and insurance). This is consistent to the observations made for the synthetically generated structures where larger graphs were exhibiting a larger improvement.The hybrid method does tend to do worst with BIC scores (used to measure divergence). This may be due to the fact that GES allows more parameters and thus can push the resulting structure closer to the distribution.

Imap

Similarly to the above section, this section shows four figures which show how the Imap algorithm performs in terms of both structure and distribution fit. In general the hybrid method performs better in terms of F-measure (except on one setting in which the F-measure for GES is better than that of Imap). Interestingly, both algorithms seem to achieve similar results for BIC scores. This suggests that with the additional constraint put on the search strategy, it is feasible to get a structure that has fewer parameters yet attains a similar distribution fit. Since this learned structure is simpler (yet has a similar distribution fit than its counterpart) it is less likely to overfit the given dataset. This issue becomes more important as the number of nodes grows (since there is more room to add edges).

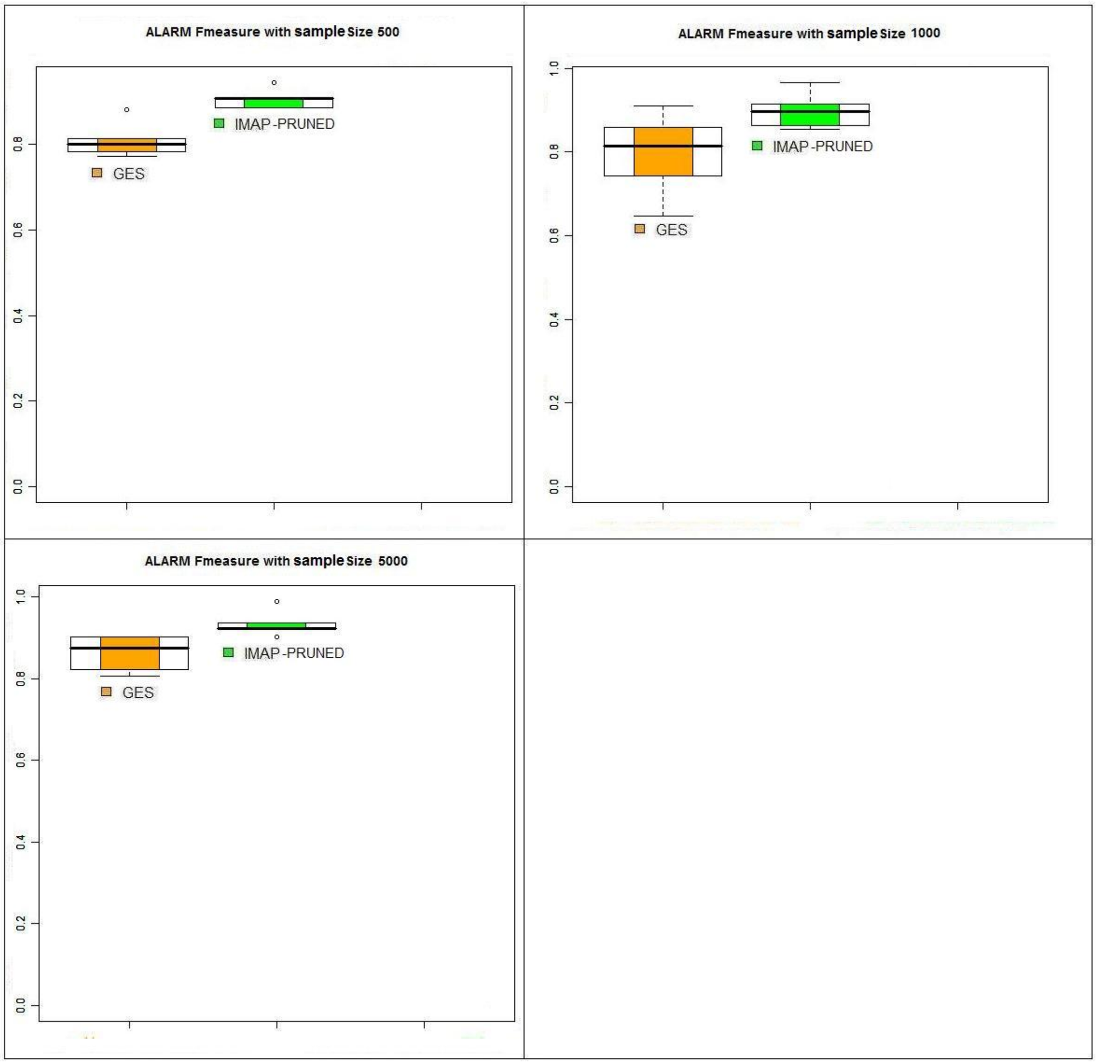


Figure 5.14: Boxplots comparing the F-measure measure in the alarm network for 3 different sample sizes. Higher F-Measure values indicate a closer fit to the target structure. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

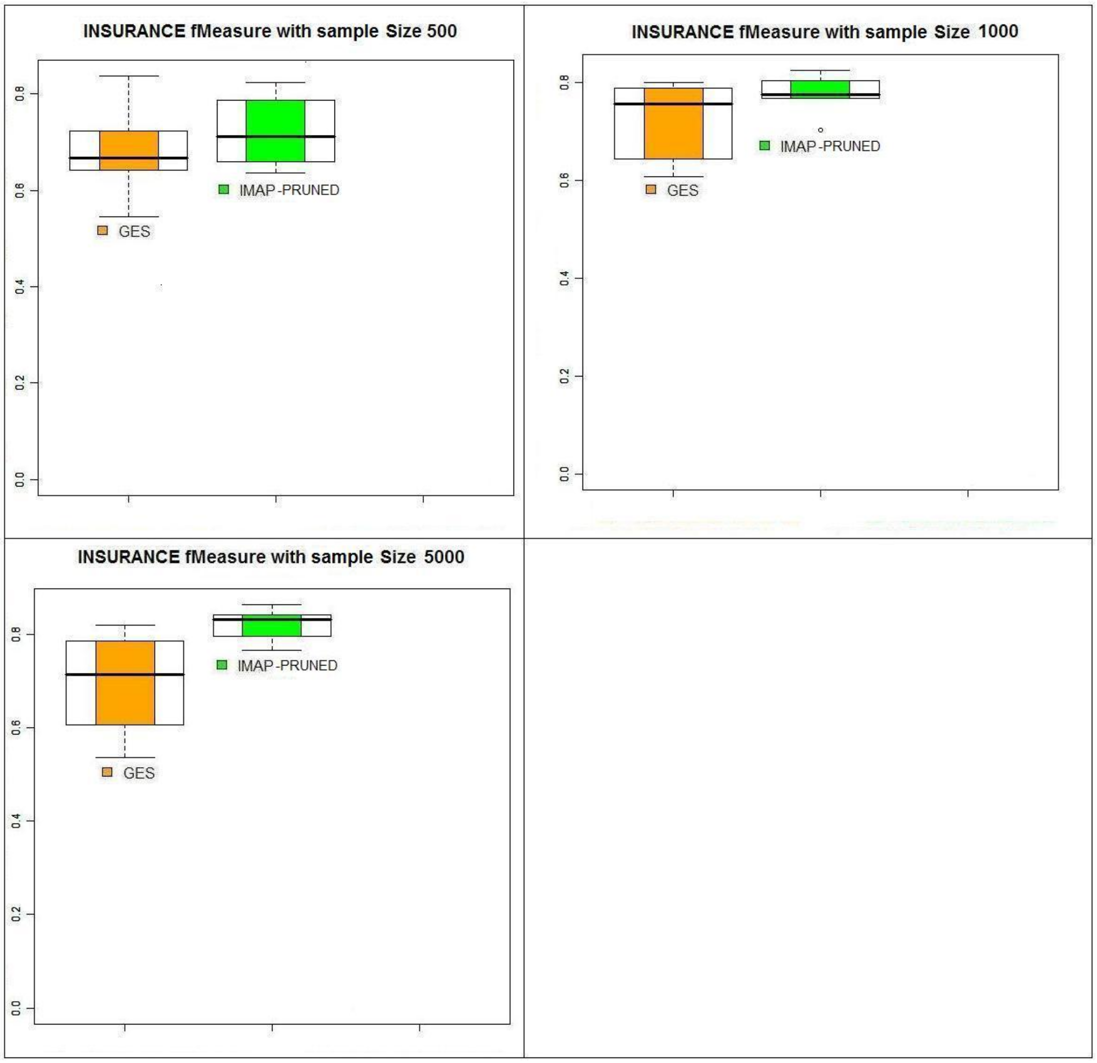


Figure 5.15: Boxplots comparing the F-measure measure in the insurance network for 3 different sample sizes. Higher F-Measure values indicate a closer fit to the target structure.

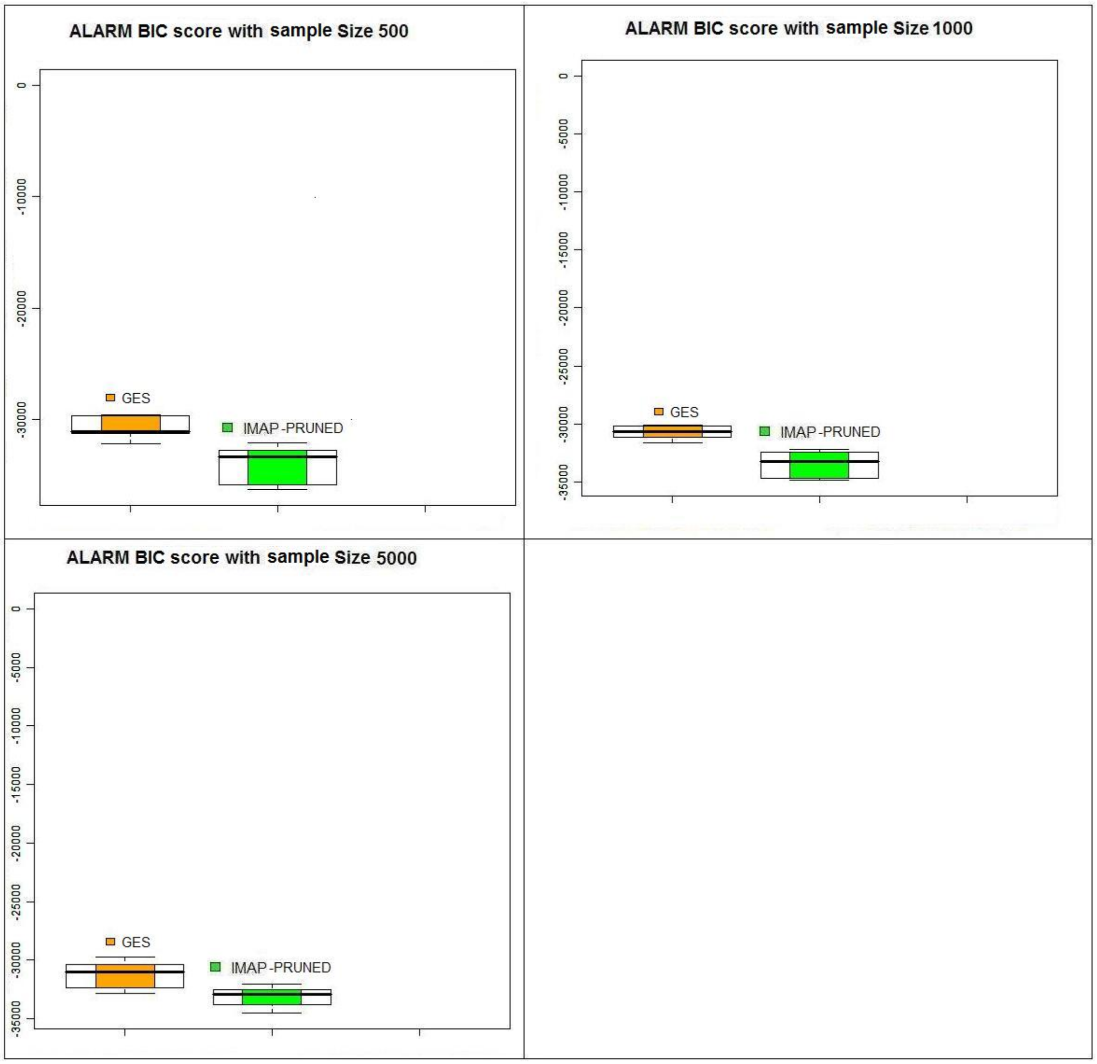


Figure 5.16: Boxplots comparing the BIC score in the alarm network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

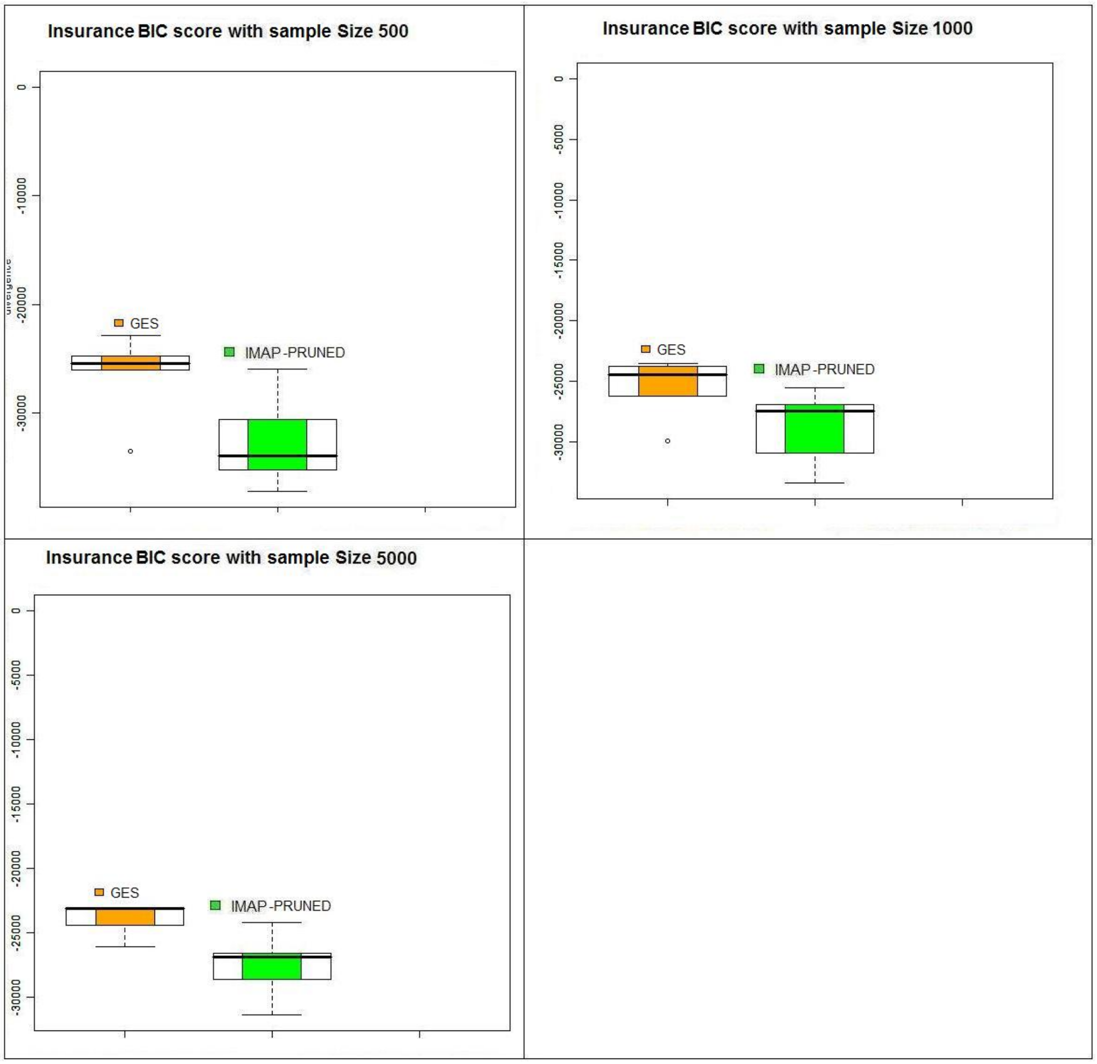


Figure 5.17: Boxplots comparing the BIC score in the insurance network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

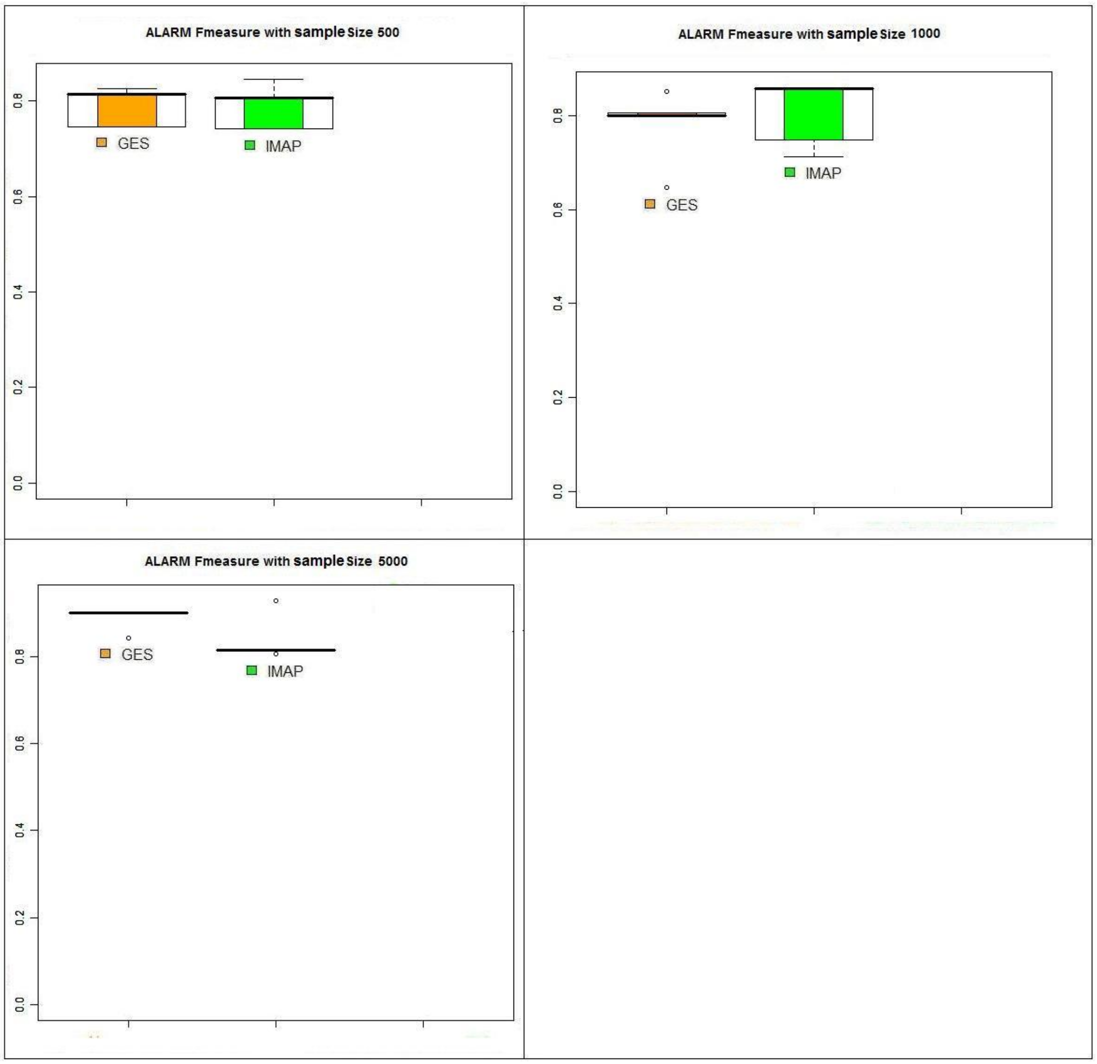


Figure 5.18: Boxplots comparing the F-measure measure in the alarm network for 3 different sample sizes. Higher F-Measure values indicate a closer fit to the target structure.

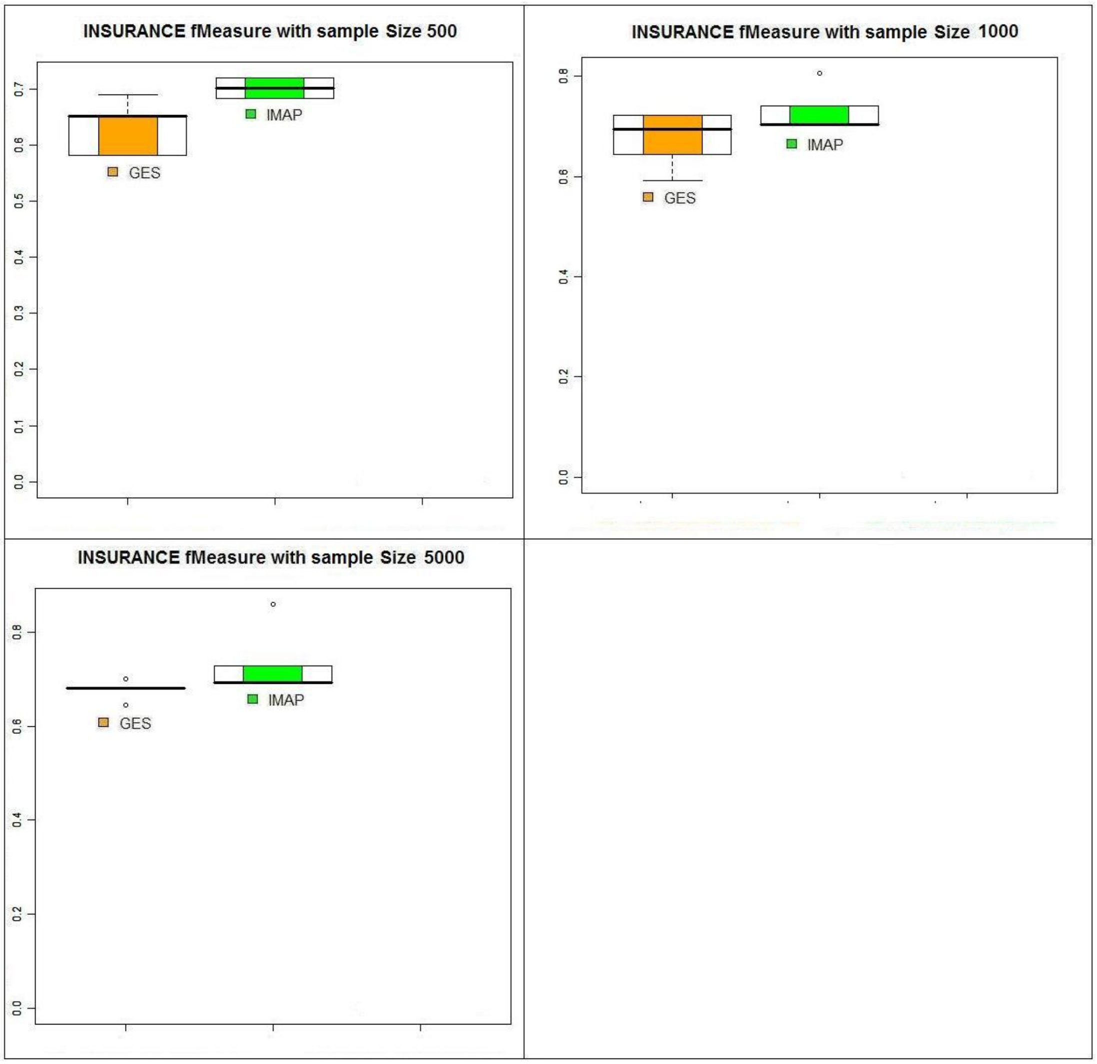


Figure 5.19: Boxplots comparing the F-measure measure in the insurance network for 3 different sample sizes. Higher F-Measure values indicate a closer fit to the target structure. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

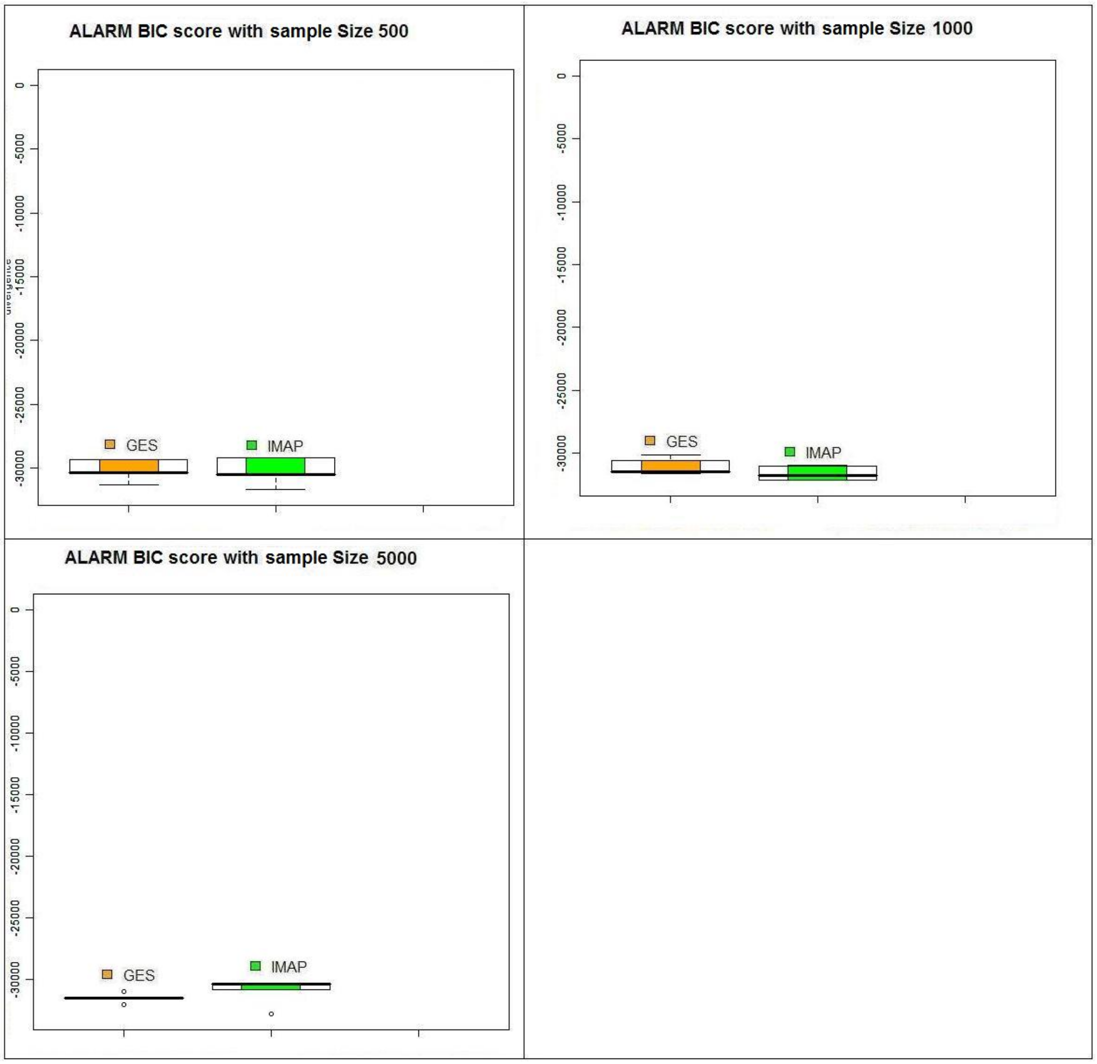


Figure 5.20: Boxplots comparing the BIC score in the alarm network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

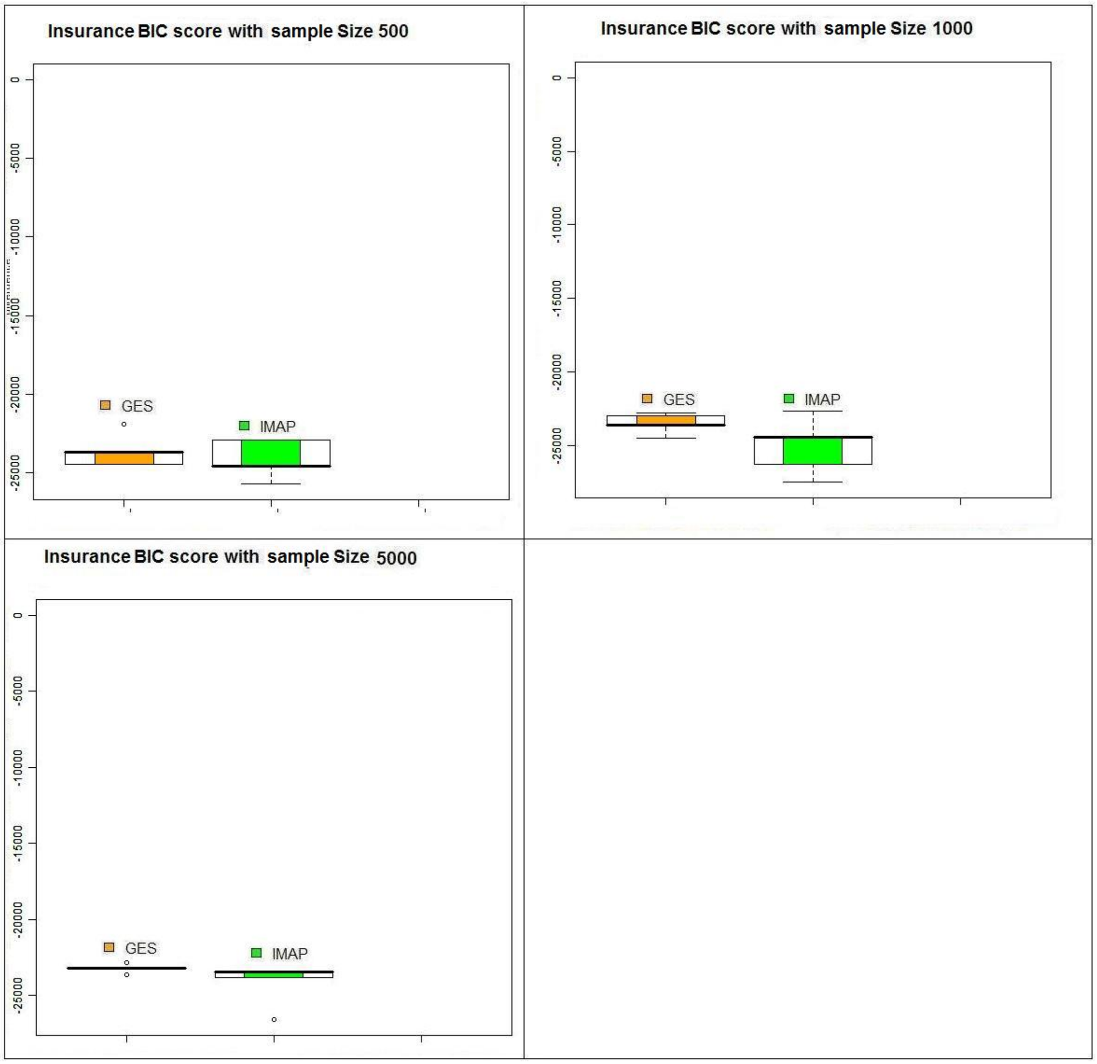


Figure 5.21: Boxplots comparing the BIC score in the insurance network for 3 different sample sizes. Lower values indicate a closer fit to the target distribution. Note: the scale for each boxplot pair in not the same. This was done in order to better display the differences for each setting.

Chapter 6

Conclusion

There are several advantages for using a hybrid learning algorithm over a non hybrid one. In this thesis, one such method was discussed in which dependency constraints were used together with a score function in order to produce a structure. The algorithms can be seen as a general way to adapt any learning algorithm based on a local neighbourhood search strategy (together with a score function) into a hybrid method which also takes into account structural properties. The experiments performed suggest that this strategy can lead to improvement in some cases. All the OS experiments used the simplest variant of this strategy in which the significance level is a parameter which fixed throughout the algorithm. One way to improve these algorithms would be to use a more sophisticated strategy for rejecting hypothesis (such as a Bonferroni correction). This may lead to a reduction in Type II error rates which may help attain better performances.

Bibliography

[1] T. Van Allen and R. Greiner. Model selection criteria for learning belief nets: An empirical comparison. In *ICML*, pages 1047–1054, 2000.

[2] I. Beinlich, H. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system. In *AIME'89*, pages 247–256, 1989.

[3] Y. Benjamini and Y. Hochberg. Controllling the false discovery rate—a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society*, 57(1):289–300, 1995.

[4] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29, 1997.

[5] Gilles Blanchard and Francois Fleuret. Occam's hammer. In Nader H. Bshouty and Claudio Gentile, editors, *Learning theory*, volume 4539 of *LNAI*, pages 112–126. COLT, Springer-Verlag Berlin Heidelberg, 2007.

[6] L. De Campos. A scoring function for learning bayesian networks based on mutual info. and cond. indep. tests. *JMLR*, pages 2149–2187, 2006.

[7] D. Chickering. Optimal structure identification with greedy search. *JMLR*, 3:507–554, 2003.

[8] D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian Networks is NP-Hard. Technical Report MSR-TR-94-17, Microsoft Research, November 1994.

[9] Randy Goebel David Poole, Alan Mackworth. *Computational Intelligence a Logical Approach.* Oxford University Press, 1998.

[10] M. H. Degroot. *Probability and Statistics.* Addison Wesley, 2 edition, 1986.

[11] Jay L Devore. *Probability and Statistics.* Thomson, 2004.

[12] Drton and Perlman. A sinful approach to bayesian graphical model selection. *Journal of Statistical Planning and Inference*, 138:1179–1200, 2008.

[13] D.M. Edwards. *Introduction to Graphical Modelling.* Springer-Verlag, 2000.

[14] John Fox. Structural equation modeling with the sem package in r. 2006.

[15] D. Heckerman, D. Geiger, , and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[16] Finn V Jensen. *Bayesian networks and decision graphs.* Springer, 2001.

[17] S. Kullback and R. Leibler. On information and sufficiency. *Ann. Math. Stat.*, 22:79–86, 1951.

[18] D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. Technical Report TR CMU-CS-99-134, school of computer science, Carnegie Mellon University, 1999.

[19] C. Meek. *Graphical Models: Selecting causal and statistical models.* PhD thesis, CMU, 1997.

[20] R. E. Neapolitan. *Learning Bayesian Networks.* Pearson Education, 2004.

[21] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kauffmann, 1988.

[22] J. Ramsey, P. Spirtes, and J. Zhang. Adjacency-faithfulness and conservative causal inference. In *UAI*, pages 401–408, 2006.

[23] R. Scheines, P. Spirtes, C. Glymour, C. Meek, and T. Richardson. *TETRAD 3:Tools for Causal Modeling.* CMU Philosopy, 1996.

[24] Monte Carlo Search, B. Baesens, M. Egmont-petersen, R. Castelo, and J. Vanthienen. Learning bayesian network classifiers for credit scoring using markov chain monte carlo search, 2002.

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

[26] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Mach. Learn.*, 65(1):31–78, 2006.

[27] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2 edition, 2005.

[28] Harry Zhang. The optimality of naive bayes. 2004.