Mach Learn
DOI 10.1007/s10994-006-6889-7

The max-min hill-climbing Bayesian network structure learning algorithm

Ioannis Tsamardinos · Laura E. Brown · Constantin F. Aliferis

Received: January 07, 2005 / Revised: December 21, 2005 / Accepted: December 22, 2005 / Published online: 28 March 2006
Springer Science + Business Media, Inc. 2006

Abstract We present a new algorithm for Bayesian network structure learning, called Max-Min Hill-Climbing (*MMHC*). The algorithm combines ideas from local learning, constraint-based, and search-and-score techniques in a principled and effective way. It first reconstructs the skeleton of a Bayesian network and then performs a Bayesian-scoring greedy hill-climbing search to orient the edges. In our extensive empirical evaluation *MMHC* outperforms on average and in terms of various metrics several prototypical and state-of-the-art algorithms, namely the *PC*, Sparse Candidate, Three Phase Dependency Analysis, Optimal Reinsertion, Greedy Equivalence Search, and Greedy Search. These are the first empirical results simultaneously comparing most of the major Bayesian network algorithms against each other. *MMHC* offers certain theoretical advantages, specifically over the Sparse Candidate algorithm, corroborated by our experiments. *MMHC* and detailed results of our study are publicly available at http://www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html.

Keywords Bayesian networks · Graphical models · Structure learning

1. Introduction

A Bayesian network is a mathematical construct that compactly represents a joint probability distribution P among a set variables $\mathcal{V}$. Bayesian networks are frequently employed for modeling domain knowledge in Decision Support Systems, particularly in medicine (Beinlich et al., 1989; Cowell et al., 1999; Andreassen et al., 1989).

Editor: Andrew W. Moore

I. Tsamardinos · L. E. Brown (✉)· C. F. Aliferis
Discovery Systems Laboratory, Dept. of Biomedical Informatics, Vanderbilt University, 2209 Garland Avenue, Nashville, TN 37232-8340
e-mail: laura.e.brown@vanderbilt.edu

I. Tsamardinos
e-mail: ioannis.tsamardinos@vanderbilt.edu

C. F. Aliferis
e-mail: constantin.aliferis@vanderbilt.edu

Learning a Bayesian network from observational data is an important problem that has been studied extensively during the last decade. One reason for this is because it can be used to automatically construct Decision Support Systems. In addition, while still controversial (see Dash, 2005 for a recent discussion) learning Bayesian networks is being used for inferring possible causal relations since, under certain conditions (Spirtes, Glymour & Scheines, 2000) the edges in the graph of a Bayesian network have causal semantics (i.e. they represent direct causal influences). For example, in bioinformatics learning Bayesian networks have been used for the interpretation and discovery of gene regulatory pathways (Friedman et al., 2000). In addition, the theory of learning Bayesian networks has deep connections with variable selection for classification (Tsamardinos & Aliferis, 2003) and has been used to design algorithms that optimally solve the problem under certain conditions (Aliferis, Tsamardinos & Statnikov, 2003b; Tsamardinos, Aliferis & Statnikov, 2003c). Finally, Bayesian network learning has been used in information retrieval (Baeze-Yates & Ribiero-Neto, 1999), natural language processing (Chapman et al., 2001), and for the analysis of a medical service's performance for management decisions (Acid et al., 2004).

The recent explosion of high dimensionality data sets in the biomedical realm and other domains, owing in part to new proteomics profiling and micro-array gene expression techniques that produce data sets with tens or hundreds of thousands of variables, has posed a serious challenge to existing Bayesian network learning algorithms. Current state-of-the-art algorithms do not reliably scale up to thousands of variables in reasonable time. In addition, improving the accuracy of the generated causal hypotheses is arguably more important now since it is being used to suggest expensive and time-consuming experiments.

Learning (the most probable a posteriori under certain conditions) Bayesian network from data is an *NP-Hard* problem (Chickering, 1996; Chickering, Meek & Heckerman, 2004). In practice, the difficulty of the problem of learning large Bayesian networks from data as perceived by the community is perhaps best captured in this relatively recent quote:

> "In our view, inferring complete causal models (i.e., causal Bayesian networks) is essentially impossible in large-scale data mining applications with thousands of variables" (Silverstein et al., 2000).

In this paper we present an algorithm, called Max-Min Hill-Climbing (*MMHC*) that is able to overcome the perceived limitations. The algorithm is able to scale to distributions with thousands of variables and pushes the envelope of reliable Bayesian network learning in both terms of time and quality in a large variety of representative domains.

In general, there are two main approaches for learning Bayesian networks from data. The search-and-score approach attempts to identify the network that maximizes a score function indicating how well the network fits the data. One such score metric is the a posteriori probability of a network $\mathcal{N}$ given the data $\mathcal{D}$ and prior knowledge K, i.e., $\arg\max_{\mathcal{N}} P(\mathcal{N}|\mathcal{D}, K)$ (Cooper & Herskovits, 1992; Heckerman, Geiger & Chickering, 1995). Algorithms in this category search the space of all possible structures for the one that maximizes the score using greedy, local, or some other search algorithm.

The second approach for learning Bayesian networks is constraint-based (Spirtes, Glymour & Scheines, 2000). Algorithms following this approach estimate from the data whether certain conditional independencies between the variables hold. Typically, this estimation is performed using statistical or information theoretic measures. The conditional independence constraints are propagated throughout the graph and the networks that are inconsistent with them are eliminated from further consideration. A sound strategy for performing conditional independence tests ultimately retains (and returns) only the statistically equivalent networks consistent with the tests.

The Max-Min Hill-Climbing (*MMHC*) algorithm can be categorized as a hybrid method, using concepts and techniques from both approaches. *MMHC* first learns the skeleton (i.e., the edges without their orientation) of a Bayesian network using a local discovery algorithm called Max-Min Parents and Children (*MMPC*), a first version of which (that did not incorporate the symmetry correction, see Section 3.3) was introduced in Tsamardinos, Aliferis and Statnikov (2003a). It then orients the skeleton using a greedy Bayesian-scoring hill-climbing search. *MMHC*'s skeleton identification phase is sound in the sample limit while the orientation phase does not provide any theoretical guarantees.

MMHC can be viewed as a particular instantiation of the Sparse Candidate algorithm (*SC*) (Friedman, Nachman & Pe'er, 1999; Friedman et al., 2000), one of the first Bayesian network learning algorithms to be successfully applied to datasets with several hundred variables. In a similar fashion to *MMHC*, Sparse Candidate constrains the search of a search-and-score algorithm: each variable X is allowed to have parents only from within a predetermined candidate parents set $C(X)$ of size at most k, where k is defined by the user. Initially, the candidate parent sets are heuristically estimated, and then hill-climbing (or other instantiations of the general algorithm) is used to identify a network that (locally) maximizes the score metric. Subsequently, the candidate parent sets are re-estimated and another hill-climbing search round is initiated. A cycle of candidate sets estimation and hill-climbing is called an iteration. Sparse Candidate iterates until there is no change in the candidate sets or a given number of iterations have passed with no improvement in the network score. Experiments with the Sparse Candidate have shown that constraining the greedy search results in significant computational gains.

However, there are three main problems with the Sparse Candidate algorithm. The first is that the estimation of the candidate sets is not sound (i.e., may not identify the true set of parents) and that it may take a number of iterations to converge to an acceptable approximation of the true set of parents. The second problem with the algorithm is that the user has to guess the parameter k, i.e., the maximum number of parents allowed for any node in the network. If the user overestimates k, the algorithm will take unnecessarily long to finish and may even be rendered intractable for large datasets. If the user underestimates k, there is a risk of discovering a suboptimal network. Finally, the parameter k imposes a uniform sparseness constraint on the network. While the connectivity of the network within a subgraph may be relatively large, the network may be quite sparse within some other area. A common parameter k for all nodes will have to sacrifice either efficiency (if it is chosen large enough to accommodate both areas) or quality of reconstruction. Unfortunately, the effects of underestimating k affect quality in a non-local fashion. As we show in our experiments, if a variable has $m > k$ parents, then not only the node will be missing at least $m - k$ parents in the reconstructed network, but the errors may propagate to the rest of the network.

Max-Min Hill-Climbing (*MMHC*) alleviates all of the three problems listed above. By learning the skeleton of the Bayesian network, *MMHC* estimates the candidate parent sets: a candidate parent of X is any other variable Y sharing an edge with X. The difference from Sparse Candidate is that this identification is performed in a sound manner (given enough sample) and without requiring the user to estimate the parameter k. Essentially the maximum number of parents k is discovered and is set individually for each variable in the network. *MMHC* is accurate enough in estimating the candidate sets that only one iteration is required, rendering *MMHC* computationally less intensive than Sparse Candidate.

To identify the skeleton, *MMHC* employs tests of conditional independence, seeking variable subsets $\mathbf{Z}$ that render a pair of variables X and Y conditionally independent. This is performed in a similar fashion to *PC* (Spirtes, Glymour & Scheines, 2000), a prototypical constraint-based algorithm. The key difference between the skeleton identification phase of

the two algorithms is that they employ different search strategies for these variable subsets. We show that *MMHC*'s strategy on average performs significantly fewer test than *PC* and thus, is computationally more efficient.

In this paper, we provide an extensive evaluation of *MMHC* against a wide cross-section of other prototypical or state-of-the-art Bayesian network learning algorithm on reconstructing several Bayesian networks employed in real decision support systems from data. This is the first study of this scope providing a valuable comparison of existing algorithms for various networks and sample sizes. Our study includes *PC* (Spirtes, Glymour & Scheines, 2000), Three Phase Dependency Analysis (Cheng et al., 2002), Sparse Candidate (Friedman, Nachman & Pe'er, 1999), Optimal Reinsertion (Moore & Wong, 2003), Greedy Equivalent Search (Chickering, 2002b) and Greedy Search. In total, 4,290 networks were learned from data using a year's single-CPU time.

MMHC is released as part of Causal Explorer 1.3 (see Aliferis et al., 2003a, for a description of Causal Explorer 1.0), a tool library of local causal discovery and Bayesian network learning algorithms. Detailed results, scripts, and code are publicly available at http://www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html to facilitate further independent analysis and comparisons with future algorithms.

In our experiments, *MMHC* outperformed on average and subject to the selected implementations all other algorithms in terms of computational efficiency except for Three Phase Dependency Analysis, *TPDA*, for sample size 5000. For three algorithms (Greedy Search, PC, and Three Phase Dependency Analysis) we counted the number of statistical calls as an implementation-independent measure of efficiency. *MMHC* performed fewer calls on average than all of the three algorithms (again, except for the Three Phase Dependency Analysis algorithm for sample size 5000). Finally, in terms of the quality of reconstruction *MMHC* outperformed on average all other algorithms (except for Greedy Equivalent Search, *GES*, for sample size 1000), as measured by the number of structural errors.

Specifically, *MMHC* outperforms Sparse Candidate even though the latter is allowed several iterations to estimate the candidate parent sets and is provided with a good estimate of the k parameter. A theoretical explanation of the results is provided. Compared to the *PC* algorithm, *MMHC*'s strategy identifies the skeleton with higher accuracy while performing fewer tests of conditional independence. Additionally, unlike *PC*, *MMHC* does not break down when the available sample size is relatively small. The counter-intuitive behavior of *PC* on small sample datasets is illustrated and explained. Against the unconstrained Greedy Search, we show that *MMHC* converges to a higher quality network by searching a significantly smaller space.

The Three Phase Dependency Analysis algorithm is polynomial to the number of variables and so it asymptotically becomes faster than *MMHC* as sample size increases. However, the quality of reconstruction is never on par with that of *MMHC*.

The Greedy Equivalent Search algorithm is guaranteed to find the maximally probable a posteriori network in the sample limit, however, it is still outperformed by *MMHC* on the finite samples we have tried. Greedy Equivalent Search is also less efficient than *MMHC*: despite the fact that it performs a greedy search, the branching factor in its search space is potentially exponential to the number of variables.

Finally, the Optimal Reinsertion algorithm has the advantage of being implemented as an anytime algorithm and also of being able to identify functions violating faithfulness; nevertheless, its quality performance is still not on par with *MMHC* within the scope of our study.

Our experimental results on *MMHC* corroborate and improve on the idea, introduced by Sparse Candidate, that constraining a (Bayesian) scoring search improves the efficiency of learning. While tests of conditional independence have been employed before by several

algorithms, *MMHC* offers new insight in the use of such tests that push the envelope of Bayesian network learning.

2. Background

We denote a variable with an upper-case letter (e.g., A, V_i) and a state or value of that variable by the same lower-case letter (e.g., a, v_i). We denote a set of variables by upper-case bold-face (e.g., $\mathbf{Z}$, $\mathbf{Pa_i}$) and we use the corresponding lower-case bold-face symbol to denote an assignment of state or value to each variable in the given set (e.g., $\mathbf{z}$, $\mathbf{pa_i}$). We use calligraphic fonts for special sets of variables such as the set of all variables considered $\mathcal{V}$. In this paper we deal with discrete probability distributions and complete datasets only (i.e., all modelled variables in all training instances obtain an observed known value).

Definition 1. Two variables X and Y are *conditionally independent given* $\mathbf{Z}$ with respect to a probability distribution P, denoted as $Ind_P(X; Y|\mathbf{Z})$, if $\forall x, y, \mathbf{z}$ where $P(\mathbf{Z} = \mathbf{z}) > 0$,

$$P(X = x, Y = y \mid \mathbf{Z} = \mathbf{z}) = P(X = x \mid \mathbf{Z} = \mathbf{z})P(Y = y \mid \mathbf{Z} = \mathbf{z})$$

or

$$P(X, Y \mid \mathbf{Z}) = P(X \mid \mathbf{Z})P(Y \mid \mathbf{Z})$$

for short. We denote and define dependence as

$$Dep_P(X; Y \mid \mathbf{Z}) \equiv \neg Ind_P(X; Y \mid \mathbf{Z})$$

Definition 2. Let P be a discrete joint probability distribution of the random variables[1] in some set $\mathcal{V}$ and $\mathcal{G} = \langle \mathcal{V}, \mathsf{E} \rangle$ be a Directed Acyclic Graph (DAG). We call $\langle \mathcal{G}, P \rangle$ a (discrete) *Bayesian network* if $\langle \mathcal{G}, P \rangle$ satisfies the Markov Condition: every variable is independent of any subset of its non-descendant variables conditioned on its parents (Pearl, 1988; Spirtes, Glymour & Scheines, 1993, 2000; Glymour and Cooper, 1999; Pearl, 2000; Neapolitan, 2003).

We denote the set of the parents of variable V_i in the graph $\mathcal{G}$ as $\mathbf{Pa}_i^{\mathcal{G}}$. By utilizing the Markov Condition, it is easy to prove that for a Bayesian network $\langle \mathcal{G}, P \rangle$ the distribution P of the variables $\mathcal{V}$ can be factored as follows:

$$P(\mathcal{V}) = P(V_1, \ldots, V_n) = \prod_{V_i \in \mathcal{V}} P\left(V_i \mid \mathbf{Pa}_i^{\mathcal{G}}\right)$$

To represent a Bayesian network the structure (i.e., the BN graph) and the joint probability distribution have to be encoded; for the latter, and according to the above equation, one is required to only specify the conditional probabilities $P(V_i = v_i \mid \mathbf{Pa}_i^{\mathcal{G}} = \mathbf{pa_j})$ for each variable V_i, each possible value v_i of V_i, and each possible joint instantiation $\mathbf{pa}_j$ of its parents $\mathbf{Pa}_i^{\mathcal{G}}$.

The graph of a network in conjunction with the Markov Condition directly encode some of the independencies of the probability distribution and entail others (see Neapolitan, 2003, pp. 70 for a definition of entailment). A graphical criterion for entailment is that of d-separation (Pearl, 1988, 2000). It is defined on the basis of blocked paths:

Definition 3. A node W of a path p is a *collider* if p contains two incoming edges into W.

[1] Variables are also interchangeably called nodes or vertices in the context of a Bayesian network.

Definition 4. A path p from node X to node Y is *blocked* by a set of nodes $\mathbf{Z}$, if there is a node W on p for which one of the following two conditions hold:

1. W is not a collider and $W \in \mathbf{Z}$, or
2. W is a collider and neither W or its descendants are in $\mathbf{Z}$ (Pearl, 1988).

Definition 5. Two nodes X and Y are *d-separated* by $\mathbf{Z}$ in graph $\mathcal{G}$ (denoted as $Dsep_{\mathcal{G}}(X; Y \mid \mathbf{Z})$) if and only if every path from X to Y is blocked by $\mathbf{Z}$. Two nodes are *d-connected* if they are not d-separated.

In Verma and Pearl (1988) it is proven that a pair of nodes d-separated by a variable set in network $\langle \mathcal{G}, P \rangle$ is also conditionally independent in P given the set. The faithfulness condition below, asserts that the conditional independencies observed in the distribution of a network are not accidental properties of the distribution, but instead due to the structure of the network.

Definition 6. If all and only the conditional independencies true in the distribution P are entailed by the Markov condition applied to $\mathcal{G}$, we will say that P and $\mathcal{G}$ are *faithful to each other* (Spirtes, Glymour & Scheines, 1993, 2000; Neapolitan, 2003). Furthermore, a distribution P is *faithful* if there exists a graph, $\mathcal{G}$, to which it is faithful.

Definition 7. A Bayesian network $\langle \mathcal{G}, P \rangle$ satisfies the *faithfulness condition* if P embodies only independencies that can be represented in the DAG $\mathcal{G}$ (Spirtes, Glymour & Scheines, 1993). We will call such a Bayesian network a *faithful network*.

Theorem 1. *In a faithful BN* $\langle \mathcal{G}, P \rangle$ (Pearl, 1988)

$$Dsep_{\mathcal{G}}(X; Y \mid \mathbf{Z}) \Leftrightarrow Ind_P(X; Y \mid \mathbf{Z})$$

We assume faithfulness of the network to learn in the rest of the paper. Because of the theorem and the faithfulness assumption, the terms d-separation and conditional independence are used interchangeably in the rest of the paper.

Notice that, there are distributions P for which there is no faithful Bayesian network $\langle \mathcal{G}, P \rangle$ (however, these distributions are "rare"; see Meek, 1995, for details). Also, there may be more than one graph faithful to the same distribution P.

Learning the structure (graph) of a Bayesian network from statistical data $\mathcal{D}$ following a distribution P is an important, on-going research problem. When learning the graph structure under certain conditions (Spirtes, Glymour & Scheines, 2000; Pearl, 2000) (that include faithfulness) one can attribute a causal interpretation to the edges of the graph $\mathcal{G}$: an edge $X \rightarrow Y$ corresponds to a direct[2] causal effect; manipulating X will affect the observed distribution of Y. Because of this property of Bayesian networks, they have been used for generating causal hypotheses, particularly in bioinformatics (Friedman et al., 2000) and developmental and cognitive psychology (Glymour, 2001).

Since there may be numerous graphs $\mathcal{G}$ such that for a specific P $\langle \mathcal{G}, P \rangle$ is a Bayesian network, several definitions are possible for the problem of learning the structure of a Bayesian

[2] Direct in this context is meant relatively to the rest of the variables in the model; for example, the direct causal relation $X \rightarrow Y$ may be rendered indirect once the model is extended to include a new mediating variable Z: $X \rightarrow Z \rightarrow Y$.

network, giving preference to inducing different structures. Here, we follow Neapolitan (2003), pp. 533:

Definition 8. Let P be a faithful distribution and $\mathcal{D}$ a statistical sample following P. The problem of learning the structure of a Bayesian network given $\mathcal{D}$ is to induce a graph $\mathcal{G}$ so that $\langle \mathcal{G}, P \rangle$ is a faithful Bayesian network.

The following theorem is utilized in most constraint-based algorithms:

Theorem 2. *In a faithful BN $\langle \mathcal{G}, P \rangle$ on variables $\mathcal{V}$ there is an edge between the pair of nodes X and Y in $\mathcal{V}$ iff $Dep_P(X; Y \mid \mathbf{Z})$, for all $\mathbf{Z} \subseteq \mathcal{V}$* (Spirtes, Glymour & Scheines, 1993).

Algorithms following the constraint-based approach estimate from the data whether certain conditional independencies between the variables hold using statistical or information-theoretic tests (Glymour & Cooper, 1999; Cheng et al., 2002). If for a pair of variables X and Y it is deemed that $Ind_P(X; Y \mid \mathbf{Z})$ conditioned on some set of variables $\mathbf{Z}$, and assuming the network to be reconstructed is faithful, then there should not be an edge between X and Y in the network according to Theorem 2. In the rest of the paper, we drop the subscripts and superscripts P and $\mathcal{G}$ when they can be inferred by the context.

3. The max-min parents and children algorithm

The Bayesian network learning algorithm presented in this paper is based on the local discovery algorithm called Max-Min Parents and Children (*MMPC*) (a version of *MMPC* was published in Tsamardinos, Aliferis and Statnikov, 2003c). The Max-Min part of the algorithm name refers to the heuristic the algorithm uses, while the parents and children part refers to its output. *MMPC* is used by *MMHC* to reconstruct the skeleton of the Bayesian network before a constrained greedy search is performed to orient the edges.

We will denote the set of parents and children of T in a graph $\mathcal{G}$ as $\mathbf{PC}_T^{\mathcal{G}}$ (not to be confused with the parents of T in $\mathcal{G}$ denoted as $\mathbf{Pa}_T^{\mathcal{G}}$). If $\langle \mathcal{G}, P \rangle$ and $\langle \mathcal{G}', P \rangle$ are two faithful Bayesian networks (to the same distribution), then for any variable T, it is the case that $\mathbf{PC}_T^{\mathcal{G}} = \mathbf{PC}_T^{\mathcal{G}'}$ (Verma & Pearl, 1990; Pearl & Verma, 1991; Tsamardinos, Aliferis & Statnikov, 2003c). Thus, the set of parents and children of T is unique among all Bayesian networks faithful to the same distribution and so we will drop the superscript and denote it simply as $\mathbf{PC}_T$. Given a target variable of interest T and statistical data $\mathcal{D}$, *MMPC* returns $\mathbf{PC}_T$, provided there is a graph faithful to the data distribution and the statistical tests performed return reliable results. Notice that, a node may be a parent of T in one network and a child of T in another, e.g., the graphs $X \leftarrow T$ and $X \rightarrow T$ may both be faithful to the same distribution. However, the set of parents and children of T, i.e., $\{X\}$, remains the same in both.

MMPC run on target T provides a way to identify the existence of edges to and from T (but without being able to identify the orientation of the edges). By invoking *MMPC* with each variable as the target one can identify all the edges (in an un-oriented fashion) in the network, i.e., identify the skeleton of the Bayesian network. To fully reconstruct the network one has to further orient the edges; this is discussed in Section 5 and gives rise to the Max-Min Hill-Climbing (*MMHC*) algorithm.

3.1. $\overline{MMPC}$

For clarity of presentation, we first describe a simplified version of the algorithm we call $\overline{MMPC}$ that may return false positives depending on the structure, i.e., it may return a superset of $\mathbf{PC}_T$. We then present the complete and sound *MMPC*.

$\overline{MMPC}$ (pseudo-code shown in Algorithm 1) invokes the function for testing $Ind(X; T|\mathbf{Z})$, that returns *true* if X and T are conditionally independent given $\mathbf{Z}$ as estimated by a statistical test on the training data $\mathcal{D}$.[3] Function $Assoc(X; T|\mathbf{Z})$ is an estimate of the strength of association (dependency) of X and T given $\mathbf{Z}$. We assume that $Ind(X; T|\mathbf{Z}) \Leftrightarrow (Assoc(X; T|\mathbf{Z}) = 0)$. Details on the functions' implementation are given in Section 4.

Definition 9. We define the minimum association of X and T relative to a feature subset $\mathbf{Z}$, denoted as $MinAssoc(X; T|\mathbf{Z})$, as

$$MinAssoc(X; T|\mathbf{Z}) = \min_{\mathbf{S} \subseteq \mathbf{Z}} Assoc(X; T|\mathbf{S})$$

i.e., as the minimum association achieved between X and T over all subsets of $\mathbf{Z}$.

Algorithm 1 $\overline{MMPC}$ Algorithm

```
1: procedure MMPC (T,D)
       Input: target variable T; data D
       Output: the parents and children of T in any Bayesian
       network faithfully representing the data distribution
       %Phase I: Forward
2:     CPC = ∅
3:     repeat
4:         ⟨F, assocF⟩ = MaxMinHeuristic(T; CPC)
5:         if assocF ≠ 0 then
6:             CPC = CPC ∪ F
7:         end if
8:     until CPC has not changed

       %Phase II: Backward
9:     for all X ∈ CPC do
10:        if ∃S ⊆ CPC, s.t. Ind(X;T|S) then
11:            CPC = CPC \ {X}
12:        end if
13:    end for

14:    return CPC
15: end procedure

16: procedure MaxMinHeuristic(T,CPC)
       Input: target variable T; subset of variables CPC
       Output: the maximum over all variables of the minimum asso-
       ciation with T relative to CPC, and the variable that achieves
       the maximum
17:    assocF = max_{X∈V} MinAssoc(X;T|CPC)
18:    F = arg max_{X∈V} MinAssoc(X;T|CPC)
19:    return ⟨F, assocF⟩
20: end procedure
```

[3] For simplicity of notation, $\mathcal{D}$ is omitted from the parameter list of all functions that use the data.

Recall that according to Theorem 2, if one identifies a subset of variables $\mathbf{Z}$ such that $Ind(X; T|\mathbf{Z})$, then there is no edge between X and T in the induced graph. For each variable X, $\overline{MMPC}$ attempts to quickly identify a conditioning set $\mathbf{Z}$ for which the independency with T holds and thus prove that X does not belong in $\mathbf{PC}_T$.

$\overline{MMPC}$ discovers $\mathbf{PC}_T$ using a two-phase scheme. In phase I, the forward phase, variables enter sequentially a candidate $\mathbf{PC}_T$, called **CPC** by use of a heuristic function. In each iteration, **CPC** is augmented according to the following:

Max-Min Heuristic: select the variable that **maximizes** the minimum association with T relative to **CPC**.

(hence the name of the algorithm). The heuristic is admissible in the sense that all variables with an edge to or from T and possibly more will eventually enter **CPC**. The intuitive justification for the heuristic is to select the variable that remains highly associated with T despite our best efforts (i.e., after conditioning on all subsets of **CPC**) to make the variable independent of T. Phase I stops when all remaining variables are independent of the target T given some subset of **CPC**, i.e., the maximum minimum association reaches zero.

In phase II, $\overline{MMPC}$ attempts to remove the false positives that may have entered in the first phase. This is achieved by testing whether $Ind(X; T|\mathbf{S})$ for some subset $\mathbf{S} \subseteq$ **CPC**. If the condition holds, X is removed from **CPC**.

In practice, the search over all subsets in Lines 10, 17, and 18 in the algorithm is limited by the available sample. In Section 4 we include a detailed discussion for the justification of the restriction and its theoretical implications, while Section 7 discusses the computational complexity ramifications.

3.2. Example trace

We now provide an example trace of $\overline{MMPC}$ with target node T shown in Fig. 1(a). We assume that the data $\mathcal{D}$ given to the algorithm are sampled from the distribution of the network in Fig. 1(a) and additionally, that the network is faithful. We assume that the sample size is large enough for the results of all tests of conditional independence performed by $\overline{MMPC}$ to return the correct result. Thus, the results of each call to $Ind(X; T|\mathbf{Z})$ can be assessed simply by looking at the graph of Fig. 1(a) and determining whether $Dsep(X; T \mid \mathbf{Z})$ or not.

In the Figs. 1(b–m) the candidate parents and children set **CPC** is the set of nodes in the dashed texture. The variable selected by the Max-Min heuristic is denoted with a circle around it.

It is easy to see that $\mathbf{S}_1 \subseteq \mathbf{S}_2 \Rightarrow MinAssoc(X; Y \mid \mathbf{S}_1) \geq MinAssoc(X; Y \mid \mathbf{S}_2)$. Thus, as the **CPC** increases the minimum association relative to **CPC** can only be reduced. For this reason, a variable that achieves zero minimum association with T (i.e., becomes independent of T given some subset of **CPC**) can never enter **CPC**, is crossed out with an X mark in the figures, and not considered again by the algorithm.

Initially (Fig. 1(b)) $\mathbf{CPC} = \emptyset$ for the target variable T and let us assume that variable A is selected by the Max-Min Heuristic. During the evaluation of the heuristic, nodes B and E are found to have a zero minimum association and are crossed out. In Fig. 1(c), A enters **CPC**. In the next iteration, D is selected by $\overline{MMPC}$'s heuristic (Fig. 1(d)). In Fig. 1(e), D enters **CPC** which now becomes $\{A, D\}$.

In the next iteration (Fig. 1(f)), variables H and J are found to be independent of T conditioned on some subset of **CPC**, namely $\{D\}$, and get crossed out and C is selected by the heuristic. It enters **CPC** in Fig. 1(g).

In the fourth iteration, I is selected (Fig. 1(h)) and enters the **CPC** (Fig. 1(i)). Finally, in Fig. 1(j) all remaining variables reach a minimum association of zero and are crossed out, at which point the forward phase completes.

The algorithm now enters the second (backward) phase. The current **CPC** is $\{A, C, D, I\}$ (Fig. 1(k)). Phase II will discover that $Ind(A; T \mid \{C, D\})$ (Fig. 1(l)) and will remove A from the **CPC** as a false positive. $\overline{MMPC}$ will return the $\mathbf{PC}_T = \{C, D, I\}$ (Fig. 1(m)).

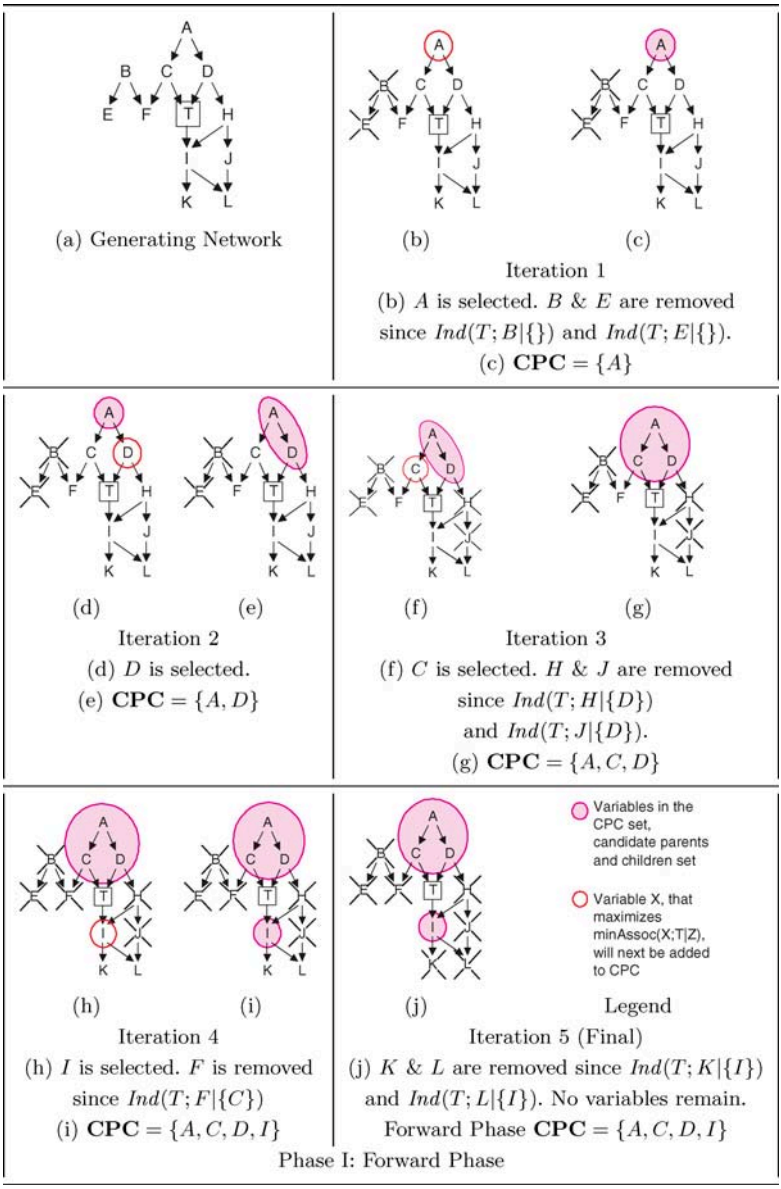


Fig. 1 Part 1: (a) Example trace of $\overline{MMPC}$, forward phase

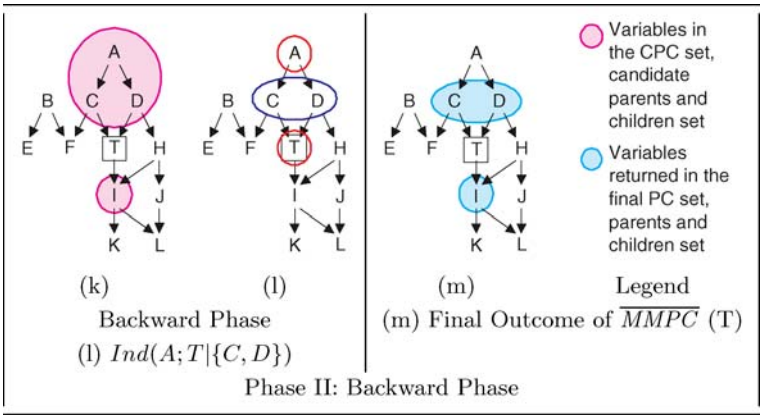
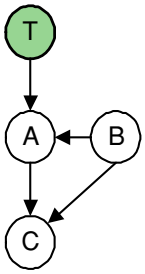


Fig. 1 Part 2: (b) Example trace of $\overline{MMPC}$, backward phase

Fig. 2 An example where $\overline{MMPC}$ will return false positives when run for target T



3.3. *MMPC*

Assuming faithfulness, $\overline{MMPC}$ will return no false negatives, i.e., will include all members of $\mathbf{PC}_T$ in its output. However, there is a case where $\overline{MMPC}$ may return false positives. This case is shown in Fig. 2. In this network, notice that C will enter and remain in the output of $\overline{MMPC}$. This is because C is dependent on T conditioned on all subsets of T's parents and children, namely both $\{A\}$ and the empty set. The problem is that by conditioning on the empty set, the path $T \rightarrow A \rightarrow C$ d-connects T and C; conditioning on $\{A\}$, the path $T \rightarrow A \leftarrow B \rightarrow C$ d-connects T and C.

The only way to d-separate T and C would be to condition on both A and B simultaneously. However, B will be removed from **CPC** since it is independent from T given the empty set. So, the algorithm will not condition on the set $\{A, B\}$.

MMPC corrects for this case and is shown in Algorithm 2. Notice that in the example of Fig. 2, even though $\overline{MMPC}(T, \mathcal{D})$ will falsely return $C \in \mathbf{PC}_T$, when run with target C, it will not include T in the output, i.e., $T \notin \mathbf{PC}_C$. Since the relation **PC** should be symmetric, this break of symmetry in the output of the algorithm is an indication of a false positive member.[4] Algorithm *MMPC* checks whether $T \in \overline{MMPC}(X, \mathcal{D})$ for all $X \in \overline{MMPC}(T, \mathcal{D})$; if this is not the case it removes X from its output.

[4] The break of symmetry may be used to direct certain edges, i.e., in our example C has to be a descendant of T (see Lemma 2) as pointed out to us by one of the anonymous reviewers.

Algorithm 2 Algorithm *MMPC*

```
1: procedure MMPC(T,D)
2:     CPC = MMPC(T, D)
3:     for every variable X ∈ CPC do
4:         if T ∉ MMPC(X, D) then
5:             CPC = CPC \ X
6:         end if
7:     end for

8:     return CPC
9: end procedure
```

In Appendix A we prove the correctness of the algorithm under the following three assumptions:

1. The distribution of the data $\mathcal{D}$ is faithful.
2. For any X and $\mathbf{Z}$, $Assoc(X; T|\mathbf{Z}) \geq 0$ with equality holding if and only if $Ind(X; T|\mathbf{Z})$.
3. The tests $Ind(X; T|\mathbf{Z})$ as estimated by the data $\mathcal{D}$ return $Ind_P(X; T \mid \mathbf{Z})$.

Theorem 3. *Algorithm MMPC*$(T, \mathcal{D})$ *will return* $\mathbf{PC}_T$.

4. Tests of conditional independence and measures of association

The algorithm is based on tests of conditional independence and measures of the strength of association between a pair of variables. We now describe how such tests are implemented.

First consider the test $Ind(X_i; X_j \mid \mathbf{X}_k)$. To implement the test we calculate the G^2 statistic as in Spirtes, Glymour and Scheines (2000), under the null hypothesis of the conditional independence holding. Let $S_{ijk}^{ab\mathbf{c}}$ be the number of times in the data where $X_i = a$, $X_j = b$ and $\mathbf{X}_k = \mathbf{c}$. We define in a similar fashion, $S_{ik}^{a\mathbf{c}}$, $S_{jk}^{b\mathbf{c}}$, and $S_k^{\mathbf{c}}$. Then, the G^2 statistic is defined as (Spirtes, Glymour & Scheines, 2000; Neapolitan, 2003)

$$G^2 = 2 \sum_{a,b,\mathbf{c}} S_{ijk}^{ab\mathbf{c}} \ \ln \frac{S_{ijk}^{ab\mathbf{c}} \ S_k^{\mathbf{c}}}{S_{ik}^{a\mathbf{c}} \ S_{jk}^{b\mathbf{c}}}.$$

The G^2 statistic is asymptotically distributed as χ^2 with appropriate degrees of freedom. Assuming no structural zeros the number of degrees of freedom is:

$$df = (\mid D(X_i) \mid \ - 1)(\mid D(X_j) \mid - 1) \prod_{X_l \in \mathbf{X}_k} \mid D(X_l) \mid$$

where $D(X)$ is the domain (number of distinct values) of variable X. As a heuristic, Spirtes, Glymour & Scheines (2000) reduce the number of degrees of freedom by one for each cell of the contingency tables of the expected (under the independence hypothesis) distribution (i.e., for each $S_{ik}^{a\mathbf{c}} \ S_{jk}^{b\mathbf{c}}$ product) that is equal to zero. In our implementation we calculate the degrees of freedom following Steck & Jaakkola (2002) instead (see calculation of Effective Number of Parameters).

The χ^2 test returns a p-value that corresponds to the probability of falsely rejecting the null hypothesis given that it is true. If the p-value is less than a significance level α (set to 0.05 in our experiments) the null hypothesis is rejected. If the independence hypothesis cannot be rejected, it is accepted instead. A more detailed discussion on this use of independence tests is in Neapolitan (2003) pp. 593.

As a measure of association, i.e., function *Assoc* in the pseudo-code, we used the negative p-value returned by the G^2 test of independence: the smaller the p-value, the higher the association. To break the ties among equal p-values we used the G^2 statistic. Again, a p value less than the 0.05 threshold is considered to indicate a zero association.

The value of G^2 requires computing the number of occurrences of all different possible patterns of variables X_i, X_j, and $\mathbf{X}_k$, i.e., $S_{ijk}^{ab\mathbf{c}}, \forall a, b, \mathbf{c}$. For example, if $\mathbf{X}_k$ contains five binary variables, and X_i and X_j are also binary, the number of possible patterns is 2^7. This implies that the number of training samples required to accurately estimate the expected value of these counts is exponential to the size of the conditioning set. Following the practice used in Spirtes, Glymour and Scheines (2000) in our experiments we do not perform an independence test (i.e., we assume independence) unless there are at least five training instances on average per parameter (count) to be estimated.

The implications of this restriction is that at Lines 10, 17, and 18 in Algorithm 1 only a limited number of subsets of **CPC** actually participate to the calls to functions *Ind* and *Assoc*. When sample is limited this method may allow false positives to be included in the output (in the worst case all other nodes). This is the case, for example, if a variable, X requires conditioning on at least two other variables to be made d-separated from T, but the available sample suffices to condition only on one other variable. Note that limiting the size of the conditioning set does not limit the number of candidate parents of a node, as in the Sparse Candidate algorithm. Rather it limits the size of the subsets of **CPC** used in the tests of conditional independence. For example, T might have ten variables in the **CPC** but the conditioning set sizes in the tests might be limited to pairwise independence with T.

In order to compute the G^2 statistic and count the possible patterns of the variables of each test, we create an array storing each count, then traverse the training sample once to compute them. The non-zero counts are identified and used in the G^2 formula. This technique is exponential to the number of variables in the conditional set and linear to the number of training cases. Better algorithms exist that only take time linear to the number of training instances, independent of the size of the conditioning set. Also, advanced data structures (Moore & Wong, 2003) can be employed to improve the time complexity.

Our implementation uses a χ^2 test to assess conditional independence and strength of association. We selected it because it is a statistical test (in contrast to the estimated mutual information for example, see Section 13), it is asymptotically correct for a general discrete multinomial distribution (e.g., it does not assume ordinal variables), and relatively easy to compute (unlike for example a permutation test).

Any other reliable, parametric or non-parametric test to assess conditional independence and strength of association could be used. For example, the Three Phase Dependency Analysis algorithm included in our experiments uses tests based on the estimated mutual information. We have conducted experiments (not reported here) with a Bayesian test of conditional independence published in Margaritis and Thrun (2001) (preliminary results were inconclusive) as well as a linear continuous conditional independence test (Fisher's z-test; see Spirtes, Glymour & Scheines, 2000, for details; preliminary results are very encouraging). Also, proper sample-power calculations could be used when deciding to accept the null hypothesis of conditional independence tests whenever distributional assumptions allow for such calculations.

5. The Max-Min Hill-Climbing algorithm

In this section, we present the Max-Min Hill-Climbing algorithm (*MMHC*) for learning the structure of a Bayesian network (Brown, Tsamardinos & Aliferis, 2004). The algorithm first

Algorithm 3 *MMHC* Algorithm

1: **procedure** MMHC($\mathcal{D}$)
Input: data $\mathcal{D}$
Output: a DAG on the variables in $\mathcal{D}$
% Restrict
2: **for** every variable $X \in \mathcal{V}$ **do**
3: $\mathbf{PC}_X$ = MMPC($X, \mathcal{D}$)
4: **end for**
% Search
5: Starting from an empty graph perform Greedy Hill-Climbing with operators *add-edge*, *delete-edge*, *reverse-edge*. Only try operator *add-edge* $Y \rightarrow X$ if $Y \in \mathbf{PC}_X$.
6: **Return** the highest scoring DAG found
7: **end procedure**

identifies the parents and children set of each variable, then performs a greedy hill-climbing search in the space of Bayesian networks. The search begins with an empty graph. The edge addition, deletion, or direction reversal that leads to the largest increase in score (the BDeu score was used) is taken and the search continues in a similar fashion recursively. *The important difference from standard greedy search is that the search is constrained to only consider adding an edge if it was discovered by MMPC in the first phase.* In our experiments, we followed the Sparse Candidate implementation (also to allow for a fair comparison) and extended the greedy search with a TABU list (Friedman, Nachman & Pe'er, 1999). The list keeps the last 100 structures explored. Instead of applying the best local change, the best local change that results in a structure not on the list is performed in an attempt to escape local maxima. This change may actually reduce the score. When 15 changes occur without an increase in the maximum score ever encountered during search, the algorithm terminates. The overall best scoring structure is then returned.

The idea of constraining the search to improve time-efficiency first appeared in the Sparse Candidate algorithm (Friedman, Nachman & Pe'er, 1999). Experiments showed that it results in efficiency improvements over the (unconstrained) greedy search. *MMHC* builds on this idea, but employs a sound algorithm (*MMPC*) for identifying the candidate parent sets.

Notice that *MMHC* initializes the candidate parents sets to the sets of parents and children. This is because, it is not always possible in general to distinguish whether X is a parent or a child of T locally, without considering other parts and variables of the network.

When the sample size is so small that only pairwise tests of independence can be performed, *MMHC* will start the greedy search with a network that contains all edges that correspond to a detectable pairwise association.

MMHC is an instance of a general Bayesian network learning algorithmic template. For example, the Max-Min Heuristic could be replaced with a different heuristic as long as it returns a variable with non-zero pair-wise association with T if such a variable exist among the remaining variables. The resulting algorithm will still be sound if assumptions 1–3 in Section 3.3 hold. In the sample limit, the effect of the heuristic is only in terms of computational efficiency: the more false positives the heuristic allows to enter **CPC**, the more computational strain will be imposed in Phase II of $\overline{MMPC}$. Other variants may include interleaving Phases I and II of $\overline{MMPC}$ in a similar fashion to the variants of the Incremental Association Markov Blanket algorithm in Tsamardinos, Aliferis and Statnikov (2003b).

An example of a successful variant of *MMPC* is the *HITON* algorithm (Aliferis, Tsamardinos & Statnikov, 2003b). *HITON* is a feature selection algorithm that first identifies the

Markov Blanket[5] of a target variable T in a similar fashion to *MMPC* by employing a heuristic based on pairwise associations only and interleaving Phases I and II. In addition, *HITON* performs a backward variable elimination wrapping phase based on Support Vector Machine classification or regression. *HITON* outperformed several other state-of-the-art variable selection algorithms on a wide variety high-dimensional (up to 140,000 variables) datasets in biomedicine.

Similarly, the greedy search-and-score method for orienting the edges could be substituted with other search methods, as long as the search can be constrained to the edges identified by *MMPC*; for example, it could be substituted by constrained versions of the Optimal Reinsertion and the Greedy Equivalent Search. Alternatively, one could use the constraint-based edge orientation criteria such as the ones employed by *PC* and the Three Phase Dependency Analysis algorithms. These criteria are sound and guarantee that if a DAG faithful to the data distribution exists, they will correctly orient the compelled edges in the sample limit. Thus, they could be used to construct a sound version of *MMHC*.

The large sample behavior however, does not guarantee better performance in practice. In anecdotal experiments we have found that the constraint-based criteria under-perform in finite sample relative to the greedy-search currently used by *MMHC* for orientation, despite their theoretical advantages. We intend to investigate the differences between the various approaches for edges orientation in the future.

6. Optimizing the computational performance

A set of optimizations improves a single call $\overline{MMPC}(T, \mathcal{D})$.

1. As mentioned, once a variable reaches a minimum association of zero with T (and gets crossed out in Figs. 1 of the example) it is not considered again by the algorithm. In our experiments most variables were eliminated in the first few iterations. Also, notice that in Lines 17 and 18 (Algorithm 1) as soon as a conditioning set $\mathbf{S}$ is discovered such that $Ind(X; T|\mathbf{S})$, there is no need to keep trying other subsets that achieve a smaller association. Similarly, in Line 10.
2. Computations between subsequent calls to the Max-Min Heuristic are shared as follows. Suppose that in iteration n variable Y is added to $\mathbf{CPC}$, so that $\mathbf{CPC}_{n+1} = \mathbf{CPC}_n \cup \{Y\}$, where the index denotes the iteration. The minimum association for any $X \in \mathcal{V}$ with T conditioned on any subset of $\mathbf{CPC}_{n+1}$ can be written as

$$\min\Big(\min_{S\subseteq \mathbf{CPC}_n} \mathbf{Assoc}(X, T \mid \mathbf{S}), \min_{S\subseteq \mathbf{CPC}_n} \mathbf{Assoc}(X, T \mid \mathbf{S} \cup \{Y\})\Big)$$

   That is, the minimum over all subsets of $\mathbf{CPC}_{n+1}$ is the minimum between the minimum achieved with all subsets that do not include the new element Y and the minimum achieved with all subsets that include Y. The first part $\min_{S\subseteq \mathbf{CPC}_n} \mathbf{Assoc}(X, T \mid \mathbf{S})$ is calculated and can be cached in iteration n. In other words, only the newly created subsets by the addition of Y need to be tested for further minimizing the association.

   A second set of optimizations shares computation among multiple calls to $\overline{MMPC}$.

[5] The Markov Blanket of a node T (Markov Boundary in Pearl's terminology) in a faithful network coincides with the set of parents, children, and spouses of T.

3. The results of the calls to $\overline{MMPC}$ are cached for use by *MMPC*.
4. If a previous call $\overline{MMPC}(X, \mathcal{D})$ has not returned T, then remove X from consideration during the invocation $\overline{MMPC}(T, \mathcal{D})$: a subset $\mathbf{Z}$ has already been found such that $Ind(X; T|\mathbf{Z})$.
5. If a previous call $\overline{MMPC}(X, \mathcal{D})$ has returned T, then it is possible X also belongs in $\mathbf{PC}_T$. As a heuristic instead of starting with an empty **CPC** during the call $\overline{MMPC}(T, \mathcal{D})$, include X in the **CPC** at Line 2 of $\overline{MMPC}$ (Algorithm 1).

None of the optimizations alters the final output (and thus the correctness) of the algorithms but they significantly improve efficiency.

7. Time complexity of the algorithms

Typically, the performance of Bayesian network-induction algorithms based on tests of conditional independence is measured in the number of association calculations and conditional independence (CI) tests executed (Spirtes, Glymour & Scheines, 2000; Margaritis & Thrun, 1999). Since in our implementation they both take exactly the same time (both calculate the p-value of the G^2 statistic) we will not differentiate between the two.

In the first phase, and when using the optimization for caching the results of the Max-Min Heuristic described in the previous section, $\overline{MMPC}$ will calculate the association of every variable with the target conditioned on all subsets of **CPC** (in the worst case). Thus, the number of tests in the first phase is bounded by $O(|\mathcal{V}| \cdot 2^{|\mathbf{CPC}|})$. In the second phase the algorithm tests for the independence of any variable in the **CPC** with the target conditioned on all subsets of the rest of the variables in the **CPC**, i.e., it performs at most $O(|\mathbf{CPC}| \cdot 2^{|\mathbf{CPC}|-1})$ tests. Thus, the total number of tests in both phases is bounded by $O(|\mathcal{V}| \cdot 2^{|\mathbf{CPC}|})$.

However, as we explained in Section 4 the size of the conditioning subsets is limited according to the available sample size and the domains of the variables. If we assume that instead of conditioning on all subsets of the **CPC** we condition on all subsets of sizes up to l, the number of tests is bound by $O(|\mathcal{V}| \cdot |\mathbf{CPC}|^{l+1})$.

In the worst case, **CPC** may grow to include all variables. The worst-case complexity is prohibitive for all but the smallest problems. However, in practice (see timing results in Section 9.2.1), we observed that the Max-Min Heuristic is powerful enough so that **CPC** is of the order of $|\mathbf{PC}|$. With this assumption, the order of the complexity becomes $O(|\mathcal{V}| \cdot |\mathbf{PC}|^{l+1})$. With caching all the calls to $\overline{MMPC}$ the overall cost of identifying the skeleton of the Bayesian network (i.e., calling *MMPC* with all targets) is $O(|\mathcal{V}|^2|\mathbf{PC}|^{l+1})$, where **PC** is the largest set of parents and children over all variables in $\mathcal{V}$.

8. Related work

The literature in Bayesian network learning is extensive. Several of the algorithms detailed here will be included in our empirical evaluation, found in Section 9. As mentioned in Section1, the three main approaches to the problem are search-and-score, constrained based, and hybrid.

Search-and-score methods search over a space of structures employing a scoring function to guide the search. Some of the standard scoring functions are Bayesian Dirichlet (specifically BDe with uniform priors, BDeu) (Heckerman,Geiger & Chickering, 1995), Bayesian Information Criterion (BIC) (Schwarz, 1978), Akaike Information Criterion (AIC) (Akaike,

1974), Minimum Description Length (MDL) (Rissanen, 1978, 1987), and K2 (Cooper & Herskovits, 1992). None has been proven to be superior and so for our experiments we selected the widely used BDeu scoring. BDeu scoring has the attractive property that it gives the same score to equivalent structures (Heckerman, Geiger & Chickering, 1995, Theorem 6), i.e., structures that are statistically indistinguishable (unlike the K2 scoring metric).

One of the most basic of the search algorithms is a local greedy hill-climbing search over all DAG structures. A basic greedy search can be augmented with methods for escaping local, sub-optimal maxima. For instance, random restarts, simulated annealing, or incorporation of a TABU list are often added to a search procedure (Chickering, Geiger & Heckerman, 1995; Bouckaert, 1995). In our evaluation we use a greedy hill-climbing search algorithm with a TABU list referred to as Greedy Search (*GS*) in Section 9 (see Section 5 for a detailed description).

The size of the search space of greedy search (i.e., the number of possible DAGs) is super-exponential to the number of variables. Two distinct approaches have emerged to improve the efficiency of these methods. The first approach reduces the complexity of the search by transforming the search space itself. For example, the Greedy Bayesian Pattern Search (*GBPS*) algorithm transforms the traditional search over DAGs to a search over equivalence classes of DAGs, called PDAGs (Spirtes & Meek, 1995) (see Section 9.1.4). The algorithm in Acid and de Campos (2003) locally searches in the space of restricted acyclic partially directed graphs (RPDAGs). One of the most prominent algorithms in this class is the Greedy Equivalent Search (*GES*) algorithm (Meek, 1997; Chickering, 2002a,b), (included in our evaluation). Greedy Equivalent Search, like greedy Bayesian pattern search (*GBPS*), also searches in the space of equivalence classes (PDAGs), however, it has the attractive property that it is guaranteed to identify in the sample limit the most probable a posteriori Bayesian network provided that the data distribution is faithful. A similar algorithm has been proposed in Kocka, Bouckaert and Studeny (2001) and a version that provides a trade-off between greediness and randomness has also been proposed (Nielson, Kocka & Pena, 2003).

The second approach to improve efficiency of the search uses constraints placed on the search. The *K2* algorithm (Cooper & Herskovits, 1992) combines the *K2* metric with a greedy hill-climbing search and requires a total variable ordering. The ordering constrains the search for parents of a node to nodes appearing earlier in the ordering. Another example is the Sparse Candidate algorithm (Friedman, Nachman & Pe'er, 1999), which only allows a variable to have a maximum of up to k parents (before the greedy search a set of candidate parents of each node is estimated). Sparse Candidate is included in our evaluation in Section 9.

The same idea using a quite different approach is taken in the Optimal Reinsertion (*OR*) algorithm (Moore & Wong, 2003) (also included in our evaluation). Starting from an initial structure a target node is chosen and all arcs into and out of the node are severed. Subject to constraints, the optimal set of arcs directed in and out of the target are reinserted. This search operator is applied repeatedly until no additional changes can be made to the DAG structure. Optimal Reinsertion makes use of specialized data-structures involving AD-search (Moore & Schneider, 2002), an extension of AD-trees (Moore & Lee, 1998; Komarek & Moore, 2000) to make tractable the evaluation of search operators. Optimal Reinsertion is one of the few algorithms that can identify parents that are connected to a node via a parity, or some other non-faithful function.

The second main approach to Bayesian network learning are constraint-based techniques. In general, constraint-based algorithms use tests of conditional independence and measures of association to impose constraints on the network structure and infer the final DAG. A prototypical constraint-based algorithm is the *PC* algorithm (from the first names of the inventors, Peter Spirtes and Clark Glymour) (Spirtes, Glymour & Scheines, 2000) which

evolved from the *SGS* (Spirtes, Glymour, Scheines) algorithm (Spirtes, Glymour & Scheines, 1990). The Three Phase Dependency Analysis (*TPDA*) algorithm (Cheng et al., 2002) follows the constraint-based paradigm and introduces the idea of using an information theoretic metric to search and test for conditional independencies. Both the *PC* and Three Phase Dependency Analysis algorithms are considered in the empirical evaluation.

Hybrid algorithms combine the search-and-score and constraint-based techniques. The first hybrid algorithm to appear is the *CB* algorithm (Singh and Valtorta, 1993). *CB* generates a node ordering using *SGS* and then invokes *K2* to orient the edges. The *PC+GBPS* algorithm (Spirtes & Meek, 1995) employs the *PC* algorithm to establish an initial pattern that is used as the basis of a search-and-score procedure. The Essential Graph Search (*EGS*) (Dash and Druzdzel, 1999) algorithm repeatedly invokes *PC* with random node orderings and thresholds. The *BENEDICT* method (BElief NEtworks DIscovery using Cut-set Techniques, in Acid and de Campos, 2001) uses the concept of d-separation and cut-sets to define a search score metric measuring the discrepancy between the graphical independence relationships and the data.

Recently an exact algorithm has been proposed that is guaranteed to identify the most probable a posteriori network (even for finite sample size) (Kovisto and Sood, 2004). This algorithm does not scale to datasets with more than a few dozen of variables. Also, an algorithm for learning Bayesian networks from sparse data using frequent sets has been explored (Goldenberg & Moore, 2004). This approach is able to scale to restricted types of networks with hundreds of thousand of variables. Other methods proposed for Bayesian network learning include variational methods (Jordan et al., 1999). Variational methods for learning the graphical model structure have mainly been applied to restrictive models (Ghahramani & Beal, 2001), run on very small networks (Friedman, 1998), or used to develop scoring functions (Ghahramani & Beal, 2003), and therefore are not included in this study.

9. Empirical evaluation

In this section, we present an extensive, comparative study among a wide range of prototypical and state-of-the-art algorithms, including *MMHC*. To the best of our knowledge this is the first study of this magnitude, with 4,290 networks induced in total using a year's single-CPU time. We examine the relative time-efficiency of the algorithms and the quality of structure identification over a range of different networks, sample sizes, and number of variables.

In this study, we sample training cases from the distributions of known networks; the algorithms are asked to reconstruct the original networks from the data. This is so that a gold-standard is known to quantify the structural quality of reconstruction. Our network collection consists of several networks employed in real decision support systems in the hope that these best represent the kind of distributions most frequently encountered in practice.

9.1. Experimental design

9.1.1. Algorithms

We selected the following algorithms for inclusion in the study:

- Sparse Candidate (*SC*, Friedman, Nachman & Pe'er, 1999)
- *PC* (Spirtes, Glymour & Scheines, 2000)
- Three Phase Dependency Analysis (*TPDA*, Cheng et al., 2002)

- Optimal Reinsertion (*OR*, Moore and Wong, 2003)
- Greedy Hill-Climbing Search (*GS*, using the BDeu scoring)
- Greedy Equivalent Search (*GES*, Chickering, 2002a)
- Max-Min Hill-Climbing (*MMHC*, our novel hybrid algorithm, Brown, Tsamardinos & Aliferis, 2004)

A more detailed description of the algorithms is provided in Section 8. While this selection includes most prototypical and state-of-the-art algorithms, it is by no means exhaustive.

In terms of specific implementations, we followed the following protocol: (1) We preferred the authors' implementation whenever available and whenever it satisfied our experimental needs, namely, could run on our experimental platform, posed no variable size limitation on the input, and output at least a minimal set of required statistics we needed for our evaluation. (2) If the above conditions were not met, we used the best publicly available implementation of the algorithm that meets the conditions. (3) Otherwise, we re-implemented the algorithm and used our version.

Specifically, we used the Sparse Candidate (*SC*) and the Optimal Reinsertion (*OR*) authors' implementation. Tetrad 4.3.1's Greedy Equivalent Search (*GES*) implementation (the authors of Greedy Equivalent Search have not released an implementation of the algorithm) and our implementations of *MMHC*, Greedy Search (*GS*), *PC* (following Spirtes, Glymour & Scheines, 2000, and Tetrad 4.3.1, re-implemented due to number of variables of input limitations), and Three Phase Dependency Analysis (*TPDA*) (re-implemented due to platform and number of variables of input limitations) in Matlab 6.5.[6] All of the Matlab algorithm implementations are available in the Causal Explorer system (Aliferis et al., 2003a). Every effort was taken for our implementations to match the originals: the output and computation effort of our versions closely match the originals in all networks the latter could accept. In reviewing the implementation of the Greedy Equivalent Search (*GES*) in Tetrad we noted that many optimizations suggested in Chickering (2002a) were not implemented, therefore the timing results for this algorithm are an upper bound.

For the Sparse Candidate (*SC*) we used the Bayesian scoring heuristic (called Score in Friedman, Nachman and Pe'er, 1999) and an equivalent sample size of 10. The maximum allowed size for the candidate parents' sets k was set to $k = 5$ and $k = 10$. The above choices follow the Sparse Candidate authors' usage and best performing parameter values in their published work (Friedman, Nachman & Pe'er, 1999; Friedman et al., 2000).

For the Greedy Search (*GS*), Greedy Equivalent Search (*GES*) and *MMHC* we used the BDeu score with equivalent sample size 10. *MMHC*'s and *PC*'s statistical threshold for the χ^2 p-values were set to the standard 5% and Three Phase Dependency Analysis' (*TPDA*) mutual information threshold was set to 1% as suggested by the authors.

Optimal Reinsertion (*OR*) requires two parameters, one which is similar to the parameter k of the Sparse Candidate (*SC*) and corresponds to the maximum number of parents allowed for a node, and the other is the maximum allowed time for the algorithm to run, (since it is an anytime algorithm). We vary the first parameter within the set $\{5, 10, 20\}$ as suggested by the authors, while the time parameter was set to be one and two times the time used by *MMHC* on the corresponding dataset (the algorithm will be labeled as *OR1* and *OR2* respectively). The scoring function used for Optimal Reinsertion (*OR*) was the BDeu scoring and the algorithm

[6] Sparse Candidate is available at http://www.cs.huji.ac.il/∼nirf/LibB.html.
Optimal Reinsertion is available at http://www.autonlab.org/autonweb/-showSoftware/149/.
Tetrad 4.3.1 with Greedy Equivalent Search is available at http://www.phil.cmu.edu/projects/tetrad/index.html.
The version of *MMHC* used in this paper is in the on-line supplement while the most up-to-date version is part of the Causal Explorer system at http://www.dsl-lab.org/sw_alg_techtransf.html.

was set to use hill climbing to optimize the final network, as in the original publication of the algorithm. The released *OR* implementation does not accept an equivalent sample size parameter and uses an internally calculated default value.

The network priors in the BDeu score of Sparse Candidate (*SC*) are calculated as in Heckerman, Geiger and Chickering (1995) where kappa $\kappa = 1/(ESS + 1)$ and *ESS* is the equivalent sample size. We follow the same practice in our implementations of the BDeu score in *MMHC* and Greedy Search (*GS*). The network priors in Greedy Equivalent Search (*GES*) are calculated following Chickering (2002a), while the method used to calculate them in the Optimal Reinsertion (*OR*) implementation is unknown.

9.1.2. Networks

The networks used in the evaluation are mainly obtained from real decision support systems that cover a wide range of real life applications, such as medicine, agriculture, weather forecasting, financial modeling and animal breeding[7]. The networks are from the following references: CHILD (Cowell et al., 1999), INSURANCE (Binder et al., 1997), ALARM (Beinlich et al., 1989), HAILFINDER (Jensen & Jensen, 1996), MILDEW (Jensen & Jensen, 1996), BARLEY (Kristensen & Rasmussen, 2002), MUNIN (Andreassen et al., 1989), PIGS (Jensen, 1997), and LINK (Jensen & Kong, 1996). We also constructed and used a Bayesian network, we call GENE using the Sparse Candidate (*SC*) algorithm on gene expression microarray data (Spellman et al., 1998) with the same procedure as in Friedman et al. (2000).

We also desired to experiment with larger networks than what is available in the public domain. In prior work (Tsamardinos et al., 2006; Statnikov, Tsamardinos & Aliferis, 2003), we have developed a method that generates large Bayesian networks by tiling several copies of smaller Bayesian networks until we reach a desired number of variables. The tiling is performed in a way that maintains the structural and probabilistic properties of the original network in the tiled network. Using this method we are able to examine the behavior of an algorithm as the number of variables increases, while the difficulty of learning the network remains the same.

Using the tiling algorithm we created versions of several of the smaller networks. The new Bayesian networks contained 3, 5, and 10 copies of the originals and are indicated with the number of tiles next to the name of a network. Information on all networks included in the study is given in Table 1.

9.1.3. Datasets

From each of the networks we randomly sampled 5 datasets with 500, 1000, and 5000 training cases each (all datasets are available on the associated web-page online). The number of training cases in a dataset is hereafter referred to as the sample size (SS). *Each reported statistic is the average over the 5 runs* of an algorithm on these different samples from the distribution of the corresponding network, unless otherwise stated.

9.1.4. Measures of performance

We employ two metrics to compare the algorithms in terms of execution speed. The first such metric is the execution time which is dependent on the specific implementation, but

[7]All networks and data are available from the on-line supplement. Most of the data sets used are also available at the Bayesian Network Repository, http://www.cs.huji.ac.il/labs/compbio/Repository.

Table 1 Bayesian networks used in the evaluation study

Network	Num. vars	Num. edges	Max In/Out-degree	Min/Max $\|PCset\|$	Domain range
Child	20	25	2 / 7	1 / 8	2–6
Child3	60	79	3 / 7	1 / 8	2–6
Child5	100	126	2 / 7	1 / 8	2–6
Child10	200	257	2 / 7	1 / 8	2–6
Insurance	27	52	3 / 7	1 / 9	2–5
Insurance3	81	163	4 / 7	1 / 9	2–5
Insurance5	135	281	5 / 8	1 / 10	2–5
Insurance10	270	556	5 / 8	1 / 11	2–5
Alarm	37	46	4 / 5	1 / 6	2–4
Alarm3	111	149	4 / 5	1 / 6	2–4
Alarm5	185	265	4 / 6	1 / 8	2–4
Alarm10	370	570	4 / 7	1 / 9	2–4
Hailfinder	56	66	4 / 16	1 / 17	2–11
Hailfinder3	168	283	5 / 18	1 / 19	2–11
Hailfinder5	280	458	5 / 18	1 / 19	2–11
Hailfinder10	560	1017	5 / 20	1 / 21	2–11
Mildew	35	46	3 / 3	1 / 5	3–100
Barley	48	84	4 / 5	1 / 8	2–67
Munin	189	282	3 / 15	1 / 15	1–21
Pigs	441	592	2 / 39	1 / 41	3–3
Link	724	1125	3 / 14	0 / 17	2–4
Gene	801	972	4 / 10	0 / 11	3–5

nevertheless captures all computations performed by an algorithm and can be defined on all present and future algorithms.

The second metric indicating computational efficiency is the total number of calls to tests of independence $Ind(X; T|\mathbf{Z})$, measures of association $Assoc(X; T|\mathbf{Z})$, and calculations of local scores (i.e., the score component of a node given its parents) $Score(T, \{X\} \cup \mathbf{Z})$. Calculating $Ind(X; T|\mathbf{Z})$, $Assoc(X; T|\mathbf{Z})$, and $Score(T, \{X\} \cup \mathbf{Z})$ can be computed at roughly the same time and several algorithms spent most of their time performing such calls. We call this metric the *number of statistical calls performed* by an algorithm and we report it only for the algorithms that we have implemented, namely Greedy Search (*GS*), Three Phase Dependency Analysis (*TPDA*), *PC*, and Max-Min Hill Climbing (*MMHC*). We remind the reader that *MMHC* calls all of the above functions, while for example pure constraint-based algorithms such as the *PC* may only use tests of independence. While the number of statistical calls is implementation independent, it can perhaps capture most but not necessarily all the different computations performed by an algorithm.

In comparing the quality of reconstruction, there are several measures proposed in the literature. One such popular metric is the BDeu score (Heckerman, Geiger & Chickering, 1995). Under certain assumptions it corresponds to the a posteriori probability (after having

seen the data) of the structure learned. It penalizes for the number of parameters (and edges) used in the network according to an arbitrary parameter called the equivalent sample size. The BDeu scores reported in this paper are calculated on a separate test set sampled from the gold-standard network containing 5000 samples. The equivalent sample size (ESS) used is 10 and the network priors are calculated as in Heckerman, Geiger and Chickering (1995) where $\kappa = 1/(ESS + 1)$ (in a similar fashion as in the implementations of *MMHC*, Greedy Search (*GS*), and Sparse Candidate (*SC*) in Section 9.1.1). Notice that, the BDeu reported is calculated in a uniform fashion across all different algorithms, even though some of them may use different network priors or equivalent sample sizes internally to learn the output network.

One of the attractive properties of the BDeu score is that it assigns the same score to the networks that capture the same set of dependencies and independencies and thus are statistically indistinguishable from observational data only. Such sets of statistically indistinguishable networks are called Markov Equivalent:

Definition 10. Two DAGs G_1 and G_2 on the same set of nodes are *Markov Equivalent* if for every three mutually disjoint subsets $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{V}$, $Ind_{G_1}(\mathbf{A}, \mathbf{B}|\mathbf{C}) \Leftrightarrow Ind_{G_2}(\mathbf{A}, \mathbf{B}|\mathbf{C})$. We will call *compelled edges* all the edges with the same orientation in all DAGs in the same equivalence class. A DAG pattern for a Markov Equivalence class, also called **PDAG**, is a graph with the same edges as the DAGs in the equivalence class, but has oriented all and only the compelled edges (Spirtes, Glymour & Scheines, 2000; Neapolitan, 2003).

The *PC* and *TPDA* algorithm returns a PDAG; to score a PDAG, a DAG extension of the PDAG is found (using the algorithm described in Dor and Tarsi, 1992) and is scored (as mentioned all such extensions have the same BDeu score).

Notice, that using the BDeu score as a metric of reconstruction quality has the following two problems. First, the score corresponds to the a posteriori probability of a network only under certain conditions (e.g., a Dirichlet distribution of the hyperparameters); it is unknown to what degree these assumptions hold in distributions encountered in practice. Second, the score depends on the equivalent sample size and network priors used. Since, typically, the same arbitrary value of this parameter is used both during learning and for scoring the learned network, the metric favors algorithms that use the BDeu score for learning. In fact, the BDeu score does not rely on the structure of the original, gold standard network at all; instead it employs several assumptions to score the networks.

For those reasons, in addition to the BDeu score we report a metric that we call the Structural Hamming Distance (*SHD*). The Structural Hamming Distance directly compares the structure of the learned and the original networks and its use is fully oriented toward discovery, rather than inference. We define the Structural Hamming Distance between two PDAGs as the number of the following operators required to make the PDAGs match: add or delete an undirected edge, and add, remove, or reverse the orientation of an edge (Algorithm 4). Thus, an algorithm will be penalized by an increase of the score by 1 for learning a PDAG with an extra un-oriented edge and by 1 for not orienting an edge that should have been oriented (more examples are shown in Fig. 3). Algorithms that return a DAG are converted to the corresponding PDAG before calculating this measure (using the algorithm in Chickering, 1995). The reason for defining the *SHD* on PDAGs instead of DAGs is so that we do not penalize for structural differences that cannot be statistically distinguished.

Algorithm 4 *SHD* Algorithm

```
procedure SHD(Learned PDAG H,True PDAG G)
    shd = 0
    for every edge E different in H than G do
        if E is missing in H then
            shd+ = 1
        end if
        if E is extra in H then
            shd+ = 1
        end if
        if E is incorrectly oriented in H then
            % This includes reversed edges and edges that are
            undirected in one graph and directed in the other.
            shd+ = 1
        end if
    end for
end procedure
```

While the *SHD* directly compares the reconstructed network with the gold standard, penalizing both for extra and missing edges, it arbitrarily penalizes errors by the same amount. Nevertheless, penalizing for structural errors has a direct intuitive interpretation. bias an algorithm towards sensitivity of identifying edges versus specificity, either by changing the BDeu equivalent sample size or the threshold in the tests of conditional independence. Thus, it would be preferable to compare the Receiver Operating Curve (Peterson,TG & Fox, 1954) of the sensitivity and specificity of an algorithm over all possible thresholds. However, calculating the Receiver Operating Curves for all algorithms to compare against is, with

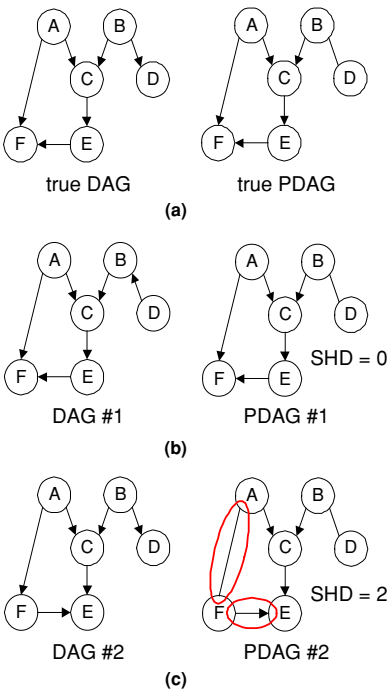


Fig. 3 Two examples of calculating Structural Hamming Distance (*SHD*). A hypothetical true DAG and its corresponding PDAG are shown in Fig. 3(a). Figure 3(b) shows a learned DAG (from data sampled from the true DAG) and its corresponding PDAG with $SHD = 0$ (notice that, the actual DAGs are different). In 3(c), PDAG #2 has a $SHD = 2$: it can match the true PDAG by adding one edge direction and by reversing another

current techniques, highly computationally expensive and was not performed as part of this study.

Another metric of quality of learning is the Kullback-Leibler divergence or KL-divergence (Kullback & Leibler, 1951) between the distribution of the reconstructed network and the original. This metric is important when the network is used for inference. However, the KL-divergence does not directly penalize for extraneous edges and parameters.[8] Moreover, in the literature, it is typically the empirical KL-divergence that is calculated (i.e., the KL-divergence as estimated from a data sample) instead of the actual KL-divergence. This is because the latter requires numerous probabilistic inferences to be made using the networks compared, each of which takes in the worst case time exponential to the number of variables.

We calculate a transformation of the KL-divergence following Acid and de Campos (2003) using the equation

$$KL(P_T, P_L) = -\mathcal{H}_{P_T}(\mathcal{V}) + \sum_{V_i \in \mathcal{V}} \mathcal{H}_{P_T}(V_i) - \sum_{V_i \in \mathcal{V}, Pa_L(V_i) \neq \emptyset} MI_{P_T}(V_i, Pa_L(V_i)), \qquad (1)$$

where P_T is the empirical joint probability distribution of the *True* network in the test set and P_L is the distribution of the *learned* network. $\mathcal{H}_P(\mathbf{Z})$ is the Shannon entropy with respect to P for the variables $\mathbf{Z}$, and $MI_{P_T}(V_i, Pa_L(V_i))$ is the mutual information between V_i and $Pa_L(V_i)$ in P_T. The Shannon entropy values do not depend upon the graph, therefore we calculate and report only the final term of the equation, following Acid and de Campos (2003). The calculations were preformed using 5000 new test cases from the network distribution. Since the last term appears with a negative sign, the results should be interpreted as follows: *the higher the value, the smaller the KL-divergence and the closer the network fits the data*. The KL-divergence results are included in the supplemental web appendices.

9.2. Results of evaluation

All algorithms were run on Pentium Xeons, 2.4 GHz, 2 GB RAM running Linux. Complete tables of all results are found in supplemental Appendices B, C, and D.[9] Some statistics are missing due to the fact that certain implementations refused to run on a specific dataset (probably due to high memory requirements) or did not complete their computations within two days running time.

9.2.1. Timing results

A summary of the timing results of the execution of all algorithms is in Table 2. We normalize the times reported by dividing by the corresponding running time of *MMHC* on the same dataset and report the averages over sample sizes and networks. Thus, a normalized running time of greater than one implies a slower algorithm than *MMHC* on the same learning task. The normalization essentially gives the same weight in the average to each network

[8] Adding extraneous edges however, may increase the KL-divergence indirectly, because the estimation of parameters becomes less accurate for a given sample.

[9] The supplemental Appendices are available on the Discovery System Laboratory web-page http://www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html

Table 2 Average normalized time results

	Sample size (SS)			
Algorithm	500	1000	5000	Average over SS
MMHC	1.00 (22/ 0)	1.00 (22/ 0)	1.00 (22/ 0)	1.00
OR1 $k = 5$	1.29 (19/ 0)	1.16 (18/ 0)	1.18 (17/ 0)	1.21*
OR1 $k = 10$	1.29 (19/ 0)	1.20 (18/ 0)	1.41 (16/ 0)	1.30*
OR1 $k = 20$	1.30 (19/ 0)	1.20 (18/ 0)	1.35 (16/ 0)	1.29*
OR2 $k = 5$	2.41 (19/ 0)	2.24 (18/ 0)	2.30 (16/ 0)	2.32*
OR2 $k = 10$	2.43 (18/ 0)	2.29 (18/ 0)	2.37 (16/ 0)	2.36*
OR2 $k = 20$	2.51 (18/ 0)	2.43 (18/ 0)	2.40 (16/ 0)	2.44*
SC $k = 5$	9.00 (21/ 0)	12.13 (22/ 0)	12.88 (18/ 2)	11.33
SC $k = 10$	10.46 (13/ 0)	14.15 (13/ 0)	15.08 (13/ 0)	13.23
GS	10.94 (20/ 2)	11.20 (20/ 2)	8.13 (20/ 2)	10.09
PC	31.68 (18/ 4)	23.81 (18/ 4)	68.57 (20/ 2)	41.35
TPDA	20.05 (21/ 1)	6.89 (21/ 1)	0.71 (22/ 0)	9.21
GES	1128.77 (7/15)	343.55 (6/16)	167.06 (6/16)	546.46

Normalized time is the running time of each algorithm for a particular sample size and network divided by the corresponding running time of *MMHC*. The first term in the parentheses is the number of networks the algorithm was averaged across and the second is the number of networks the algorithm hit the 2 day time threshold and was stopped. If those numbers do not add to 22 the algorithms refused to run on the remaining ones. Average normalized time values smaller than one correspond to an algorithm with faster running times than *MMHC*. The *reminds the reader that the Optimal Reinsertion (*OR*) algorithms running time was set to be 1 or 2 times that of *MMHC*. Also, recall the *GES* implementation did not contain all optimizations described by the original authors.

irrespective of their size, both in terms of sample and variables, but also difficulty of learning; without it, the average would be dominated by the largest or hardest to learn networks. The first number in the parentheses in the table indicates the total number of networks included in an average, while the second denotes the number of networks on which the algorithm hit the 2 day time ceiling. Those results are excluded from the average so that the results are a lower bound on the performance of *MMHC* versus the other algorithms on these datasets. Some of the provided implementations of the algorithms refused to run on several datasets due to memory limitations or unknown reasons explaining why the numbers in the parentheses do not add to 22. The complete timing results can be found in the online supplemental Appendix B.

No conclusion should be drawn on the timing results of Optimal Reinsertion (*OR*) since, as an anytime algorithm, it was instructed to run for 1 (*OR1*) and 2 (*OR2*) times the running time of *MMHC* on the corresponding dataset. *OR* periodically checks whether the time limit imposed has been exceeded and may actually take longer than the preset time to terminate.

Execution Time Conclusions. For every algorithm and sample size, *MMHC* is faster on average (subject to the selected implementations). As we can see, both other constraint-based algorithms, namely *PC* and Three Phase Dependency Analysis (*TPDA*) face problems when the sample is low (500) and perform over 30 and 20 times slower than *MMHC* on average.

Table 3 Average normalized number of statistical calls performed

	Sample size (SS)			
Algorithm	500	1000	5000	Average over SS
MMHC	1.00 (22)	1.00 (22)	1.00 (22)	1.00
GS	2.94 (20)	2.54 (20)	1.55 (20)	2.34
PC	21.33 (18)	4.94 (18)	1.38 (20)	9.22
TPDA	2.41 (21)	1.33 (21)	0.42 (22)	1.38

Normalized number of statistical calls is the number of statistical calls performed by each algorithm (number of tests of conditional independence and/or number of calls to the scoring function) for a particular sample size and network divided by *MMHC*'s calls on the same dataset. The term in parentheses is the number of networks the algorithm was averaged across. Average normalized values greater than one correspond to an algorithm performing more statistical calls than *MMHC*.

Sparse Candidate (*SC*) is only about 10 times slower at low sample, but the difference increases with higher sample, as *MMHC* is able to assess independencies more accurately. That is, for sample size 5000 Sparse Candidate (*SC*) becomes more than 15 times slower. *TPDA* improves its performance and it is the only algorithm that is faster than *MMHC* at 5000 sample.

9.2.2. Statistical calls results

Table 3 contains the normalized number of statistical calls performed for the algorithms we have implemented (excluding the cases where the 2 day limit was reached). The results analytically can be found in the online supplemental Appendix B.

The execution time and the number of statistical calls of an algorithm are not strictly proportional for various reasons. First, different implementations may use different data structures for representing and performing graph operations that affect execution time. Second, while the number of statistical calls is indicative of the computational efforts spent, typically the effort to compute a test of independence or score also depends on the size of the conditioning or parent set. Thus, not only the number, but the average size of the conditioning set determines the actual computational effort spent. Finally, all algorithms perform additional operations, such as checks for acyclicity or storing and retrieving SepSets (for the *PC* algorithm), with different frequency and of varying difficulty.

Statistical Calls Conclusions. In general, *MMHC* performs the least number of statistical calls on average, except for sample size 5000 against the *TPDA* algorithm.

9.2.3. Bayesian score results

The Bayesian score results are summarized in Table 4 (analytical results are in online Appendix C). We normalize the scores reported by dividing by the corresponding score of *MMHC* on the same dataset and report the averages over sample sizes and networks. Normalized scores are positive numbers; normalized scores less than one indicate a learned network that is more a posteriori probable (under the scoring assumptions) than that learned by *MMHC* on the same dataset. The term in the parentheses is the number of networks an algorithm was averaged across (the algorithm finished within two days).

Table 4 Average normalized bayesian score results

Algorithm	Sample size (SS) 500	1000	5000	Average over SS
MMHC	1.000 (22)	1.000 (22)	1.000 (22)	1.000
OR1 $k = 5$	1.016 (19)	1.019 (18)	1.021 (17)	1.019
OR1 $k = 10$	1.014 (19)	1.016 (18)	1.023 (16)	1.018
OR1 $k = 20$	1.016 (19)	1.022 (18)	1.022 (16)	1.020
OR2 $k = 5$	1.005 (19)	1.009 (18)	1.011 (16)	1.008
OR2 $k = 10$	1.000 (18)	1.009 (18)	1.003 (16)	1.004
OR2 $k = 20$	1.009 (18)	1.010 (18)	1.006 (16)	1.009
SC $k = 5$	0.999 (21)	0.996 (22)	1.016 (18)	1.004
SC $k = 10$	1.010 (13)	1.016 (13)	1.014 (13)	1.013
GS	0.976 (20)	0.983 (20)	0.991 (20)	0.984
PC	1.275 (18)	1.234 (18)	1.195 (20)	1.235
TPDA	1.406 (21)	1.367 (21)	1.206 (22)	1.326
GES	1.083 (7)	1.007 (6)	1.118 (6)	1.070
Empty Graph	1.498 (22)	1.520 (22)	1.546 (22)	1.521
True Graph	0.977 (22)	0.990 (22)	1.007 (22)	0.991

We normalize the scores reported by dividing by the corresponding score of *MMHC* on the same dataset. Normalized scores are positive numbers; *normalized scores less than one indicate a learned network that is more a posteriori probable (under the scoring assumptions) than that learned by MMHC on the same dataset*. The term in the parentheses is the number of networks an algorithm was averaged across (the algorithm finished within two days).

Bayesian Score Conclusions. Overall, *MMHC* is on par, or better than all algorithms. The only algorithm that outperforms *MMHC* on average is the Greedy Search (*GS*). Optimal Reinsertion (*OR*) and Sparse Candidate (*SC*) obtain approximately the same scores on average. The constraint-based algorithms, *PC* and Three Phase Dependency Analysis (*TPDA*), both have worse normalized scores (although both show improvement with as the sample size increases). Notice that Greedy Search (*GS*) is outperforming even the score of the data-generating network (denoted by True Graph in the table) showing signs of overfitting.

9.2.4. Structural hamming distance results

A summary of the Structural Hamming Distance results is presented in Table 5 (analytical results in online supplemental Appendix C). We normalize the *SHD* reported by dividing by the *SHD* of *MMHC* on the same learning task. The term in parentheses is the number of networks the algorithm was averaged across. Average normalized *SHD* values greater than one correspond to learned structures with more structural errors than those reconstructed by *MMHC*. For the CHILD and PIGS networks, *MMHC* finds the network exactly with no errors reported. In this case, the ratio of the *SHD* is undefined and was not included in taking the average over all networks (thus favoring all other algorithms, which did have errors on these networks).

Structural Hamming Distance Conclusions. *MMHC* outperforms all other algorithms for all sample sizes and networks, except for Greedy Equivalent Search (*GES*) at sample size 1000. Optimal Reinsertion (*OR*), Sparse Candidate (*SC*), and the Greedy Equivalence Search (*GES*) are the only algorithms that on average report structural errors within a 50% of *MMHC*'s *SHD*. It is worth noting that *MMHC* outperforms Sparse Candidate (*SC*) even for the GENE network (for sample sizes greater or equal to 1000), despite the fact that GENE was

Table 5 Average normalized structural hamming distance results

Algorithm	Sample size (SS) 500	1000	5000	Average over SS
MMHC	1.00 (22)	1.00 (22)	1.00 (22)	1.00
OR1 $k = 5$	1.30 (19)	1.45 (18)	1.70 (17)	1.48
OR1 $k = 10$	1.29 (19)	1.37 (18)	1.78 (16)	1.48
OR1 $k = 20$	1.31 (19)	1.45 (18)	1.86 (16)	1.54
OR2 $k = 5$	1.18 (19)	1.33 (18)	1.66 (16)	1.39
OR2 $k = 10$	1.19 (18)	1.34 (18)	1.65 (16)	1.39
OR2 $k = 20$	1.22 (18)	1.34 (18)	1.71 (16)	1.42
SC $k = 5$	1.13 (21)	1.28 (22)	1.57 (18)	1.33
SC $k = 10$	1.18 (13)	1.28 (13)	1.35 (13)	1.27
GS	1.62 (20)	2.08 (20)	1.86 (20)	1.85
PC	8.85 (18)	10.07 (18)	2.82 (20)	7.25
TPDA	9.63 (21)	10.22 (21)	1.76 (22)	7.21
GES	1.18 (7)	0.94 (6)	1.19 (6)	1.10

Normalized Structural Hamming Distance (*SHD*) is the *SHD* of each algorithm for a particular sample size and network divided by *MMHC*'s *SHD* on the same sample size and network. The term in parentheses is the number of networks the algorithm was averaged across. Average normalized *SHD* values greater than one correspond to an algorithm with more structural errors than *MMHC*.

constructed from data using Sparse Candidate (*SC*) and thus, exactly matches its inductive bias. The performance of *MMHC* improves with increased sample size relative to all other algorithms (and in absolute terms) except for PC, Three Phase Dependency Analysis (*TPDA*), and the Greedy Equivalence Search (*GES*).

9.2.5. Simultaneous comparison of time and quality

In this section we explore the trade-off between quality and time efficiency of each algorithm. An algorithm may choose to sacrifice quality to gain time efficiency; similarly, by searching larger portions of possible structures, an algorithm trades off time for the possibility of increasing quality.

Figures 4–6 plot the logarithm of the normalized *SHD* versus the logarithm of the normalized time for sample sizes 500, 1000, and 5000 respectively. Each point in the graphs represent the performance in those two metrics of a given algorithm on reconstructing a specific network. Since we are using normalized measures *MMHC*'s performance always falls on point (1,1).

A normalized *SHD* less than one means fewer structural errors than *MMHC* on the same task. Thus, any points in the graphs falling below the horizontal dashed line where $SHD = 1$ indicate cases where *MMHC* was dominated in quality by some other algorithm. Similarly, points on the left of the vertical line where $Time = 1$ indicate cases where *MMHC* was less efficient than some other algorithm. The points in the gray area correspond to learning tasks where *MMHC* is dominated both in terms of quality of reconstruction and time efficiency.

Table 6 summarizes the above results: overall, there are 5 instances of an algorithm on a particular network outperforming *MMHC* in both time and quality out of a possible 792 cases (approximately 0.63% of the time, the 792 cases from comparing *MMHC* against the 12 other algorithms over 22 networks and 3 samples sizes, $12 \times 22 \times 3 = 792$). In the rest of the cases *MMHC* outperforms the other algorithms in at least one metric.

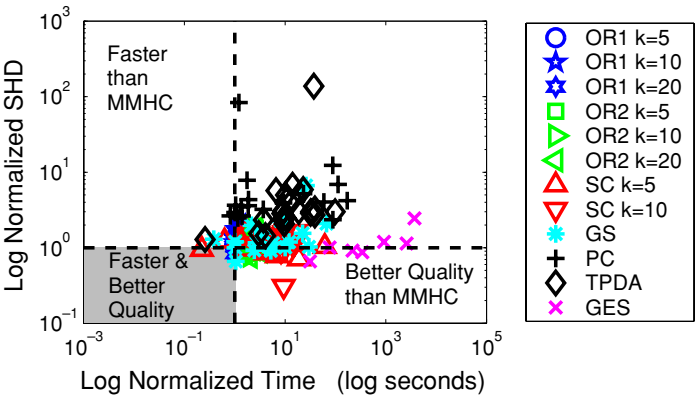


Fig. 4 Normalized time vs. normalized *SHD* at sample size 500

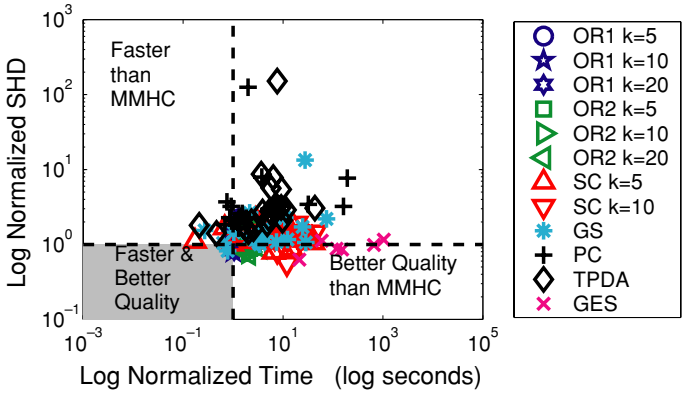


Fig. 5 Normalized time vs. normalized *SHD* at sample size 1000

Conclusions. *MMHC* typically outperforms all other algorithms in terms of both metrics. A detailed analysis of the timing results also reveals that in most cases that *MMHC* is not as efficient as some other algorithm is in some of the smallest networks (e.g., CHILD) where the constant of the time-complexity becomes more important.

For sample size 500 (Fig. 4), there is only one instance (1/264–0.38%) of an algorithm that outperforms *MMHC* in terms of both time and quality. That is Sparse Candidate (*SC*) $k = 5$ on the MUNIN network. For sample size 1000, (Fig. 5), there is again one such case (1/264–0.38%): Greedy Search (*GS*) on the CHILD network. For sample size 5000, (Fig. 6), there are three such cases (3/264–1.14%): Sparse Candidate (*SC*) $k = 5$ and Greedy Search (*GS*) on HAILFINDER, and Three Phase Dependency Algorithm (*TPDA*) on the ALARM5 network.

MMHC is never dominated in both metrics by *PC*, Sparse Candidate (*SC*) $k = 10$, and the Optimal Reinsertion (*OR2*) algorithms for any network or sample size.

In Tables 2 and 5 we showed that *MMHC* outperforms on average all other algorithms in terms of time and Structural Hamming Distance. Figures 4–6 show pictorially that *MMHC*'s average statistics are not inflated due to a few outliers where the algorithm performs especially well relatively to the other algorithms. Additional tables reporting normalized median results

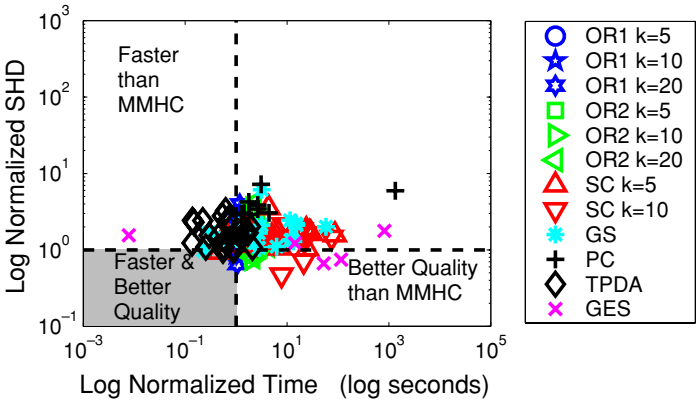


Fig. 6 Normalized time vs. normalized *SHD* at sample size 5000

Table 6 Summary of time and structural hamming distance (*SHD*) results

	Number of learning tasks where reconstruction by *MMHC* was		
Sample size	Slower	Less accurate	Slower and less accurate
500	6 / 264 (2.27%)	35 / 264 (13.26%)	1 / 264 (0.38%)
1000	13 / 264 (4.92%)	22 / 264 (8.33%)	1 / 264 (0.38%)
5000	36 / 264 (13.64%)	16 / 264 (6.06%)	3 / 264 (1.14%)
Total	55 / 792 (6.94%)	73 / 792 (9.22%)	5 / 792 (0.63%)

and graphs with error bars are also included in online Appendices B and C to provide a better sense of the distribution of the timing and quality of learning results.

9.2.6. Large sample

To explore the timing behavior of the algorithms as sample increases we ran a limited set of experiments with datasets comprising of 20,000 records. To reduce the computation time required, we selected three representative networks out of the 22 network candidates to experiment with. The three networks whose average reconstruction time (over all algorithms for sample size 5000) was closer to the average reconstruction time of all networks for sample size 5000 were selected. These are the CHILD5, CHILD10, and ALARM5 networks. To further reduce computation time the set of parameters for the *OR* and the *SC* algorithms were set to the values that exhibited the best results at sample size 5000. Table 7 summarizes the results of these computational experiments (only one dataset for each network of size 20,000 was sampled).

Large Sample Conclusions. The execution time, number of statistical calls, and quality of learning of *MMHC* remain competitive versus all other algorithms as the sample size grows. The algorithm that seems to benefit the most in terms of computational effort from the increased sample is *TPDA*, while *GS* and *SC* are still slower than *MMHC*. In terms of

Table 7 Normalized time, statistical calls, and structural hamming distance for averaged on the CHILD5, CHILD10, and ALARM5 Networks for Sample Sizes 500, 1000, 5000, 20000

Algorithm	SS = 500	SS = 1000	SS = 5000	SS = 20000
	Normalized time			
MMHC	1.00	1.00	1.00	1.00
Best OR1	1.12	1.18	1.12	1.07
Best OR2	2.21	2.14	2.13	2.07
Best SC	6.64	10.46	17.35	28.34
GS	9.18	9.13	11.10	15.92
PC	30.46	0.97	0.83	0.96
TPDA	15.37	5.77	0.62	0.51
	Normalized number of statistical calls			
MMHC	1.00	1.00	1.00	1.00
GS	3.86	3.41	2.91	2.65
PC	50.36	0.82	1.05	1.12
TPDA	3.10	1.49	0.57	0.42
	Normalized structural hamming distance			
MMHC	1.00	1.00	1.00	1.00
Best OR1	1.46	1.46	1.60	2.98
Best OR2	1.25	1.37	1.29	2.69
Best SC	1.07	1.06	0.94	1.26
GS	1.48	1.56	1.52	1.97
PC	3.16	2.45	1.54	1.87
TPDA	5.24	5.03	2.25	2.39

quality of reconstruction *MMHC*'s difference in performance from sample size 5000 to 20000 grows even larger showing a better utilization of the additional sample.

9.2.7. Scaling to thousands of variables

As a proof-of-concept experiment illustrating the ability of *MMHC* to scale up to thousands of variables we created a tiled version of the ALARM network with approximately 5000 variables (135 tiles, 4995 variables) and 6845 edges and sampled 5000 instances from its distribution. *MMHC* reconstructed the tiled ALARM-5000 in approximately 13 days total time. The reconstructed network had 1340 extra edges (specificity 99.9%), 1076 missing edges (sensitivity 84%), and 1468 wrongly oriented edges. Compare this to ALARM and ALARM10 where the reconstruction had an average of 0.8 and 27.2 extra edges and 0.8 and 109.0 missing edges respectively. The skeleton reconstruction phase took approximately 19 hours and the rest was spent on edge orientation. The sensitivity and specificity of skeleton reconstruction are calculated as in Tsamardinos et al. (2003a): sensitivity is the number of correctly identified edges over the total number of edges and specificity is the number of correctly identified non-edges over the total number of non-edges.

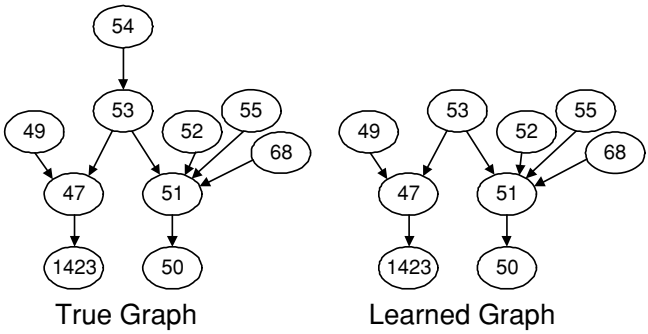


Fig. 7 Comparison of the True and Learned Networks around Node 53 in a tiled Alarm network with 5000 variables from 5000 samples. Node 53 was chosen as a representative example because the sensitivity and specificity in learning its Markov Blanket is the closest to the average sensitivity and specificity of learning Markov Blankets over all nodes

To get an intuitive sense of the accuracy of reconstruction of the network, we graph the true and the learned network around a prototypical node (with index 53) in Fig. 7. The neighborhoods of the graphs around a radius of two edges away from node 53 are plotted. Node 53 was chosen as a representative example because the sensitivity and specificity in learning its Markov Blanket is the closest to the average sensitivity and specificity of learning Markov Blankets over all nodes (the selection process is similar to the one used in Tsamardinos et al., 2003a).

Additional evidence of the scalability of *MMHC* is found in a previously reported experiment in Tsamardinos et al. (2003a). *MMPC* was run on each node and reconstructed the skeleton of a tiled-ALARM network with approximately 10,000 variables from 1000 training instances using the same hardware as above in 62 hours of single-CPU time (the task was completed in approximately 16 hours of real time by parallelizing the skeleton identification algorithm using 4 CPUs). In this experiment the sensitivity and specificity are found to be 81% (2572 missing edges out of 13640) and 99.9% (11068 extra edges) respectively. To our knowledge these are the largest unrestricted Bayesian Networks reconstructed so far (see Goldenberg and Moore, 2004, for reconstructing larger networks but of restricted form).

9.2.8. Relative performance of the algorithms

In the results above, we calculate the normalized averages of an algorithm $\mathcal{A}$ based on the set of networks on which $\mathcal{A}$ terminated within the two day limit. This allows a direct comparison with *MMHC* that terminated on all networks using all available data points. However, the reader should exercise caution when comparing the averages of two algorithms $\mathcal{A}$ and $\mathcal{B}$ with each other as they are possibly calculated on different sets of networks. To facilitate performance comparisons among any pair of algorithms, we present in on-line Appendix D the same results as above calculated on the intersection of networks of a given sample size on which all algorithms terminated (typically 10 to 12 networks out of 22 candidates). From this intersection we excluded *GES* because it causes the intersection to drop to a size of 3 networks. The results in Appendix D are consistent with the conclusions drawn in the previous sections.

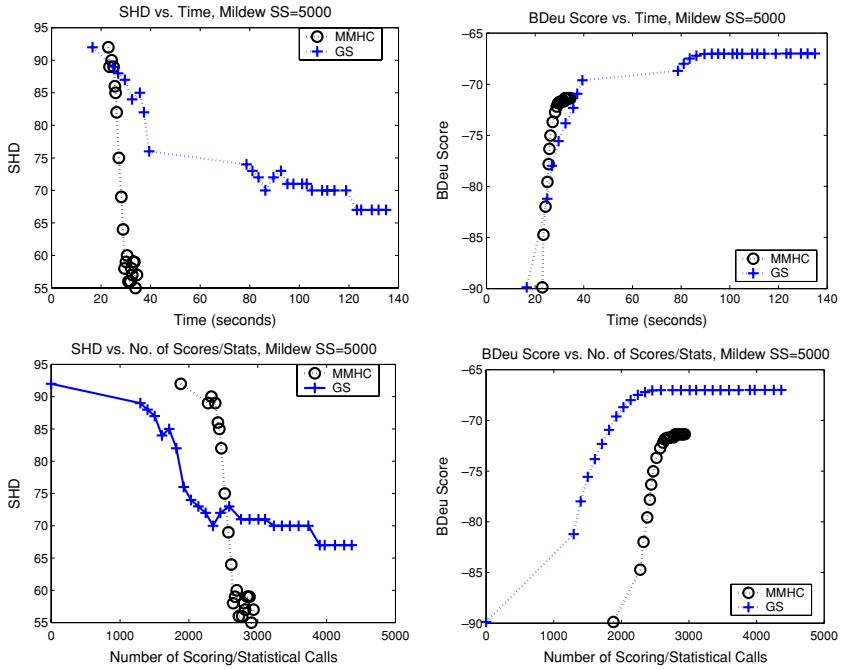


Fig. 8 Performance of *MMHC* and *GS* on the MILDEW networks. The *SHD* and BDeu score are plotted versus running time and number of statistical/scoring calls made for both algorithms

10. Greedy search vs. MMHC

In this section, we show that *MMHC*'s strategy of constraining the search, results in computational savings, particularly in sparse and large networks. This analysis is similar to the one performed for the Sparse Candidate (Friedman, Nachman & Pe'er, 1999).

We selected three representative networks on which we graphically compare the two algorithms. To select the networks, the average normalized ratio of the number of statistics for *GS* and for sample size 5000 was calculated over all networks. The three networks closest to this average were chosen (MILDEW, CHILD3, HAILFINDER10). The *SHD* and the BDeu score are plotted against time for one run (sampling) of both Greedy Search and *MMHC*.[10] In addition, the *SHD* and the BDeu score are plotted versus the number of statistical calls. The three representative cases are shown in Figs. 8–10.

Note that the search procedure and the scoring function implementations are common between the two algorithms; they differ only in that *MMHC* includes a calculation of candidate parent sets by invoking *MMPC* for each node.

Each data point plotted is sampled from when an edge is added, deleted or reversed in the current DAG. In the first column of each figure, the *SHD* is plotted versus running time (top) or number of statistics (bottom) for each network. The second column of the figure plots the BDeu score versus the running time (top) or the number of statistics (bottom).

[10] The time of these results does not match the ones reported in supplemental Appendix B, because of the additional calculations of the *SHD* and the BDeu scoring metrics for every search operator application.

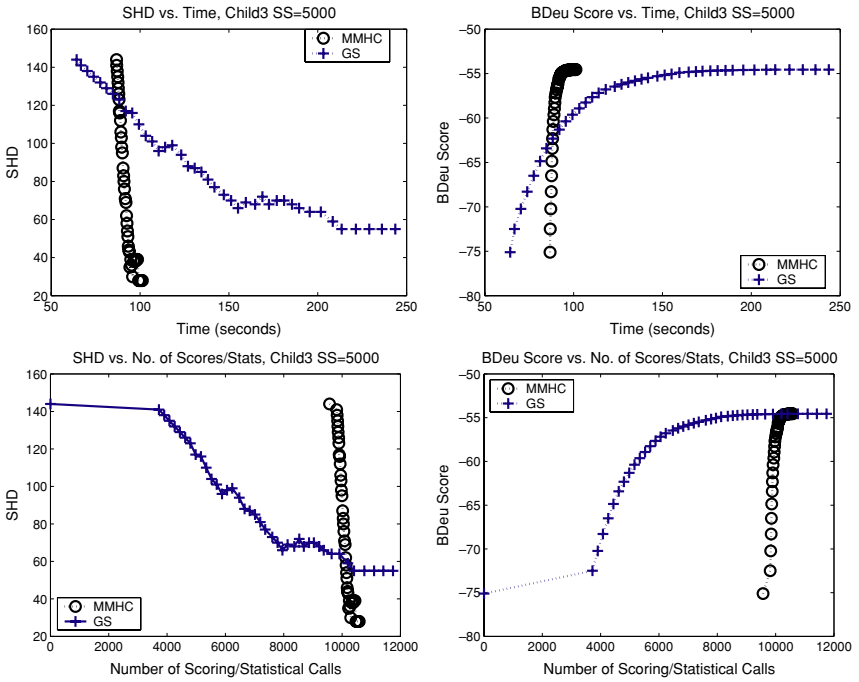


Fig. 9 Performance of *MMHC* and *GS* on the CHILD3 networks. The *SHD* and BDeu score are plotted versus running time and number of statistical/scoring calls made for both algorithms

Greedy Search exhibits an initial lag before the first search operator application because initially it calculates the scores for each possible edge addition; the subsequent score calculations are incremental and consequently much more efficient. *MMHC* also has an initial lag taking the time to run *MMPC* on each variable.

The graphs illustrate the sharp learning curve for *MMHC* to reconstruct the network once the candidate parent sets are determined. *MMHC* is shown to quickly reach a minimum *SHD* (first column) or a maximum BDeu score (second column). The overhead of constraining the search by calculating candidate parent sets results in large computational savings overall. For example, on the CHILD3 network, *MMHC* may take longer to initiate the search, but once it does, a minimum *SHD* (maximum BDeu score) is achieved rapidly.

It is interesting to note that, while Greedy Search (*GS*) may find a higher scoring network (see graph on MILDEW), the graphical structure of the DAG is not closer to the true graph as measured by *SHD*. The non-correspondence of *SHD* to the BDeu score is also shown by the fact that the *SHD* curve is non-monotonic in several instances.

A theoretical analysis of the empirical results now follows. Let us ignore for simplicity the edge reversals and deletions of Greedy Search. Then, to add k parents to a node T Greedy Search will perform $(n-1)+(n-2)+\cdots+(n-k)$ calls to the scoring function, where $n=|V|$ (we also ignore the fact that some of the scores may not be computed due to acyclicity constraints) which is equal to $kn-\frac{k(k-1)}{2}$. Thus, for node T Greedy Search will perform statistical calls on the order of kn.

In contrast, *MMHC* on that node will in the worst case perform $(n-1)+2^0(n-2)+\cdots+2^{pc-2}(n-pc)$ calls to function *Assoc*, where $pc \geq k$ is the size of the parents and children set of T. In the best case, all other nodes will be removed after the first iteration of

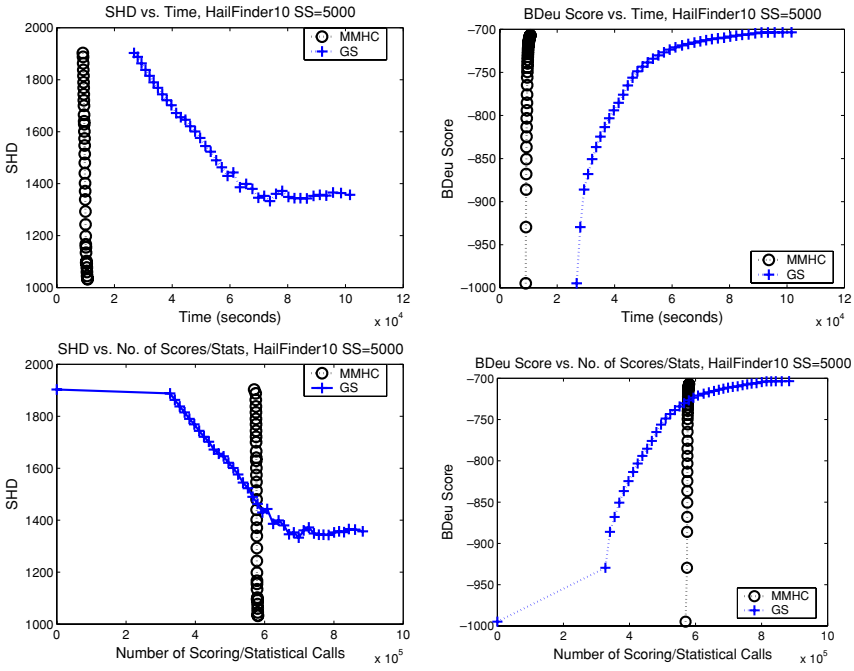


Fig. 10 Performance of *MMHC* and *GS* on the HAILFINDER10 networks. The *SHD* and BDeu score are plotted versus running time and number of statistical/scoring calls made for both algorithms

MMPC on T and the number of calls to *Assoc* will be $(n-1)+2^0(pc-1)+\cdots+2^{pc-2}1$ which is equal to $(n-1)+\sum_{i=0}^{pc-1}(pc-1-i)2^i=(n-1)+2^{pc}-(pc+1)$, which is equal to $(n-pc-2)+2^{pc}$. In addition, *MMHC* will perform $k\cdot pc-\frac{k(k-1)}{2}$ scores to add the k parents out of the pc candidates in the edge orientation phase for a total of $(n-pc-2)+k\cdot pc-\frac{k(k-1)}{2}+2^{pc}$ statistical calls, which is of the order of $n+2^{pc}$.

We see that Greedy Search considers for addition to the parents set a variable again and again even if it may never show signs of increasing the score. In contrast, *MMHC* is typically able to remove from consideration a variable after the first couple of iterations, as soon as it discovers a d-separating set from T. Most variables in sparse networks have small such sets even in cases where the parents and children sets are large (e.g., in a tree-like structure the d-separating sets have size 1 independent of the number of children).

In general, if the connectivity of a network remains the same (i.e., the number pc of the parents and children set) as n is increasing, MMHC's performance will also be improving over the Greedy Search. For $pc=10$, $k=5$, and $n=500$ and according to the above equations the two algorithms will perform about

$$n+2^{pc}=1024+500=1524<nk=500\cdot 5=2500$$

statistical calls to add k parents to some node, i.e., *MMHC* will perform about 40% less calls. When n increases to 1000 *MMHC* performs about 60% less calls.

The above statement is corroborated by our experiments. Let us consider a series of tiled networks, e.g., ALARM, ALARM3, ALARM5, and ALARM10. In each such series the

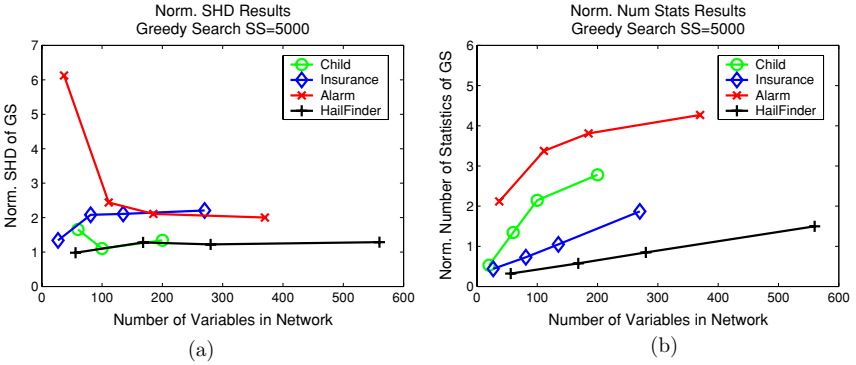


Fig. 11 Relative performance of *GS* on the four series of tiled networks: ALARM, CHILD INSURANCE, and HAILFINDER. Each series comprises of the original network, and 3, 5, and 10 tiled versions of it. (a) Normalized Structural Hamming Distance plotted against the number of variables in the networks. (b) Normalized number of statistical calls performed plotted against the number of variables in the networks. While the relative learning performance of *GS* remains the same as the number of variables increases (a), the relative number of statistical calls increase with increased problem size (b)

difficulty in learning the network and its connectivity remains the same by construction while the number of variables increases. This is graphically depicted in Fig. 11(a) where the normalized Structural Hamming Distance of Greedy Search is plotted versus the number of variables for each series: the relative quality of learning the networks remains the same as the size increases (with an outlier for ALARM).

However, the number of statistical calls is greatly affected by the network size. In Fig. 11(b) we plot the normalized number of statistical calls of Greedy Search versus the number of variables for each series of tiled networks and sample size 5000. As we can see, in some of the small networks Greedy Search performs fewer statistical calls than *MMHC* (normalized number of calls is less than 1). As the number of variables grow, Greedy Search performs more and more calls relative to *MMHC*. On-line Appendix E contains similar curves for all other algorithms, sample sizes, and performance metrics.

11. PC vs. MMHC

In this section, we show that *MMHC* is not a trivial extension of *PC*, where the edge orientation phase has been replaced with a search-and-score procedure. Instead, we show that the two algorithms employ different strategies for identifying d-separating subsets and reconstructing the skeleton.

Table 8 shows the number of association calculations and conditional independence (CI) tests (completed calls to *Ind* and *Assoc*) per variable averaged out over all networks for both *MMHC* and *PC*. The final column of this table also presents the ratio of the number of CIs performed by *PC* compared to that for *MMHC*. The average ratio is 24.2, 5.7, and 1.5 for the sample sizes of 500, 1000, and 5000 respectively. Over all the networks and sample sizes there are 30 cases where *PC* performs fewer statistical tests of *Ind* and *Assoc* than *MMHC*. In each of these cases, *MMHC* returns a higher quality network.

There are two main differences in the strategy for performing independence tests between the two algorithms. The first one is the Max-Min Heuristic employed by *MMHC* versus the heuristic employed by *PC* (see Heuristic 3, Spirtes, Glymour and Scheines, 2000, pp. 90).

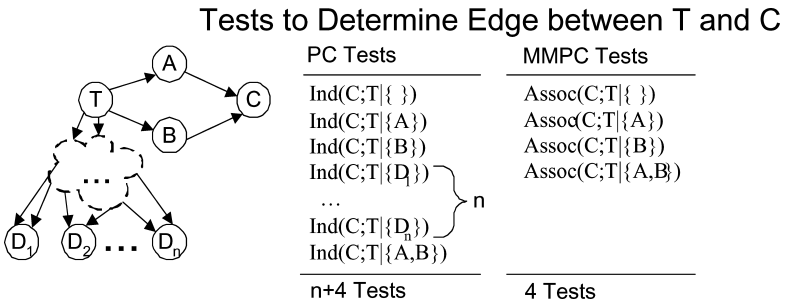


Fig. 12 An example network illustrating the different strategies for performing tests of independence and measures of strength of association between *PC* and *MMHC*. We assume nodes D_i require conditioning on at least two nodes to be d-separated from T and that both algorithms are equipped with perfect heuristics for this particular case. The column on the left is the list of tests of the form $Ind(C; T|\mathbf{S})$, for some subset $\mathbf{S}$ performed by *PC* until it removes edge $T \leftrightarrow C$ from further consideration (and the output skeleton). Notice that the tests are ordered by the size of the conditioning set. The column on the right is the list of measures of associations of the form $Assoc(C; T|\mathbf{S})$ performed by *MMHC* to remove the same edge

PC's heuristic decides which conditional test to perform next based on pairwise associations only, not conditional ones.

The second main difference is that *PC* exhausts all possible tests with conditioning set size n, before performing any test with conditioning set size $n + 1$. In contrast, *MMHC* does not restrict itself to perform the tests in order of increasing conditioning set size.

Here follows an example illustrating the above argument. In Fig. 12 a network is shown where variable T is connected to node C via nodes A and B. In addition, the structure is such that nodes $D_1, \ldots, D_n$ can only be d-separated from T by conditioning on at least two other variables. Let us concentrate on the independence tests and measures of association of the form $Ind(C; T|\mathbf{S})$ and $Assoc(C; T|\mathbf{S})$, where $\mathbf{S}$ is some subset of variables, the algorithms attempt to remove the edge $T \leftrightarrow C$ from the skeleton. We will assume that both algorithms are equipped with optimal heuristics i.e., will perform the least number of tests possible given their search strategies.

PC will first perform the test $Ind(C; T|\emptyset)$ (and all other tests with conditioning set size of zero). It will then perform the tests $Ind(C; T|A)$, $Ind(C; T|B)$ and $Ind(C; T|D_i)$ for all $i = 1, \ldots, n$.[11] All these tests will fail. Next, it will perform the test $Ind(C; T|A, B)$, which will succeed, and the edge will be removed. Thus, edge $T \leftrightarrow C$ requires *PC* a total of $n + 4$ tests of the form $Ind(C; T|\mathbf{S})$ to be removed from the skeleton.

MMHC will call $\overline{MMPC}(T, \mathcal{D})$ to determine whether there should be an edge $T \leftrightarrow C$ in the output. $\overline{MMPC}(T, \mathcal{D})$ will first calculate $Assoc(C; T|\emptyset)$ during the evaluation of the

Table 8 Average Number of Statistical Tests per Variable for the *MMHC* and *PC* algorithms at the Sample Sizes of 500, 1000, and 5000. Ratio of Number of Statistical Tests per Variable performed by *MMHC* and *PC*

Sample size	MMHC	PC	Ratio PC/MMHC
500	147.4	3048.8	24.2
1000	179.5	851.0	5.7
5000	948.1	1200.6	1.5

[11] This is because all edges $T \leftrightarrow D_i$ are still in the graph at this point, since we assumed no D_i can be d-separated from T conditioned on only one node.

Max-Min Heuristic in the first iteration. **CPC** will then be increased by the variable selected by the heuristic, e.g., A. The algorithm will then calculate $Assoc(C; T|A)$ during the second iteration and increase **CPC** by one more variable, say B.[12] The next evaluation of the Max-Min Heuristic will invoke $Assoc(C; T|B)$ and $Assoc(C; T|A, B)$, at which point variable C becomes independent of T and is never considered again by the algorithm (optimization 1 Section 6). The total number of tests of the form $Assoc(C; T|\mathbf{S})$ to remove the edge $T \leftrightarrow C$ for the call $\overline{MMPC}(T, \mathcal{D})$ is 4. When calling $\overline{MMPC}(C, \mathcal{D})$, and provided the optimization (4) in Section 6 is in effect, T should already have been removed from consideration by the algorithm. Thus, in total *MMHC* performs 4 conditional measures of association calculations versus the $n + 4$ tests required by *PC*.

The above discussion illustrates the difference between the *PC* and the *MMHC* approaches in reconstructing the skeleton. *PC* focuses on a specific conditioning set size before moving on to consider larger sizes. *MMHC* focuses on a specific variable before considering the next one.

11.1. A theoretical analysis of behavior of the *PC* for low sample cases

While the previous section explains the performance difference between the *PC* and *MMHC*, this section focuses on the difference in quality of reconstruction.

Constraint-based methods have been criticized for performing poorly when the available sample is relatively low. As shown in our experiments, *PC* indeed exhibits such behavior (see Dash and Druzdzel, 2003, for a discussion and a suggested solution based on a robust test of conditional independence), especially in networks, such as MUNIN, where the domains of certain variables are large relative to the available sample. We now provide an explanation of this behavior. We show that this property of *PC* is not representative of all constraint-based methods; for example *MMHC* has excellent performance in low sample.

PC first identifies the skeleton of a Bayesian network and then orients the edges. It starts with a complete graph, and then removes an edge X—Y if and when a subset $\mathbf{S}$ is found such that $Ind_P(X; Y \mid \mathbf{S})$. A subtle but important detail of *PC* is that the choice to start with a complete graph (i.e., it assumes that any pair of variables is dependent by default) in combination with the details of how the conditional tests are implemented *introduces a non-linearity relative to the available sample in the results of the conditional independence tests*. This observation is elaborated below.

Like many constraint-based algorithm, *PC* does not perform a test if there is not enough sample: a test $Ind_P(X; Y \mid \mathbf{Z})$ is not performed unless there are at least 10 data samples per different possible patterns of X, Y, and $\mathbf{Z}$ (e.g., if $\mathbf{Z}$ contains two binary variables, and X, Y also binary at least $2^4 \times 10$ samples are required). This is necessary because a large enough conditioning set will always render the χ^2 test unable to reliably reject the independence hypothesis.

Now, assume that X and Y are actually dependent given the empty set. When the available sample is low, *PC* will not be able to perform a test of independence and so it will assume dependence (since it starts with a complete graph). As sample increases *PC* will perform the test $Ind_P(X; Y \mid \emptyset)$, but if the association is low enough relatively to the available sample, the independence hypothesis will not be rejected by the χ^2 test and it will be accepted instead. As the available sample increases, the test will be able to reject the independency hypothesis and retain the edge. The example is presented pictorially in Fig. 13.

[12] Notice that the test $Assoc(C; T|\emptyset)$ will not be repeated when optimization (2) in Section 6 is in effect.

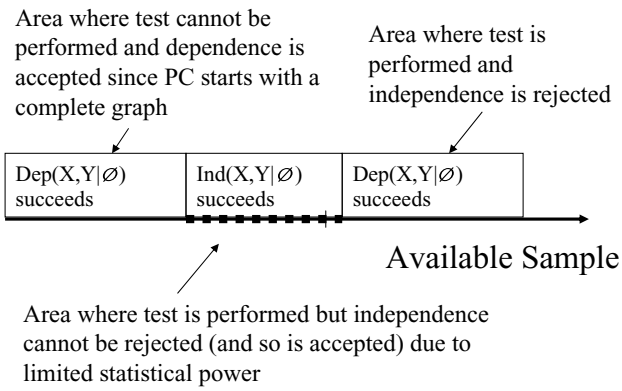


Fig. 13 An example of the non-linear behavior of the *PC* algorithm. X and Y are dependent; the results of the test $Ind_P(X; Y \mid \emptyset)$ non-linearly depend on the available sample

The consequences of such non-linearities may be catastrophic. If a variable X has a large domain size relative to the available sample no test of independence that contains X can be performed; in that case *PC* will output a network where X has an edge to every other variable. It also still blindly considers X during computations and slows down significantly. *MMHC* on the other hand does not exhibit such non-linearities because in essence it starts with an empty graph, only inserting variables into candidate parents and children sets when a dependency has been established.

Finally, we also observe that the edge orientation phase of *PC* is similarly impaired. Suppose that there is no available sample to perform any test of independence that includes variable X. Then, as discussed above, X will be connected to any other variable. In the orientation phase, for any triplet of nodes $Y - X - Z$, where Y and Z are not connected to each other, X is considered as a potential collider by *PC*. X does not belong in the SepSet(Y, Z) or any other SepSet since we cannot include the node in any test (see description of the algorithm, Spirtes, Glymour & Scheines, 2000), and so *PC* will orient $Y \rightarrow X \leftarrow Z$. In other words, for a variable X with a high domain count (relative to the available sample), all other nodes will have an edge pointing to X (except for nodes also connected to all other ones).

Again, this problem is not inherent to constraint-based methods, but a peculiarity of the specific choices of the *PC* algorithm regarding the use of the results of the statistical tests.

12. A theoretical analysis of the sparse candidate

Similarly to *MMHC*, Sparse Candidate (*SC*) first estimates for each variable X, a candidate set of parents $C(X)$ (in *MMHC* this is the parents and children set as returned by *MMPC*) of at most size k, where k is provided by the user. It then performs a constrained greedy hill-climbing search in the space of all Bayesian networks starting from the empty network. Once hill-climbing reaches a local maximum, the candidate sets are re-estimated and hill-climbing search is performed again. The original Sparse Candidate paper (Friedman, Nachman & Pe'er, 1999) provides several heuristic methods for estimating the candidate sets $C(X)$. We will call the cycle of candidate sets estimation and hill-climbing an iteration. Sparse Candidate iterates

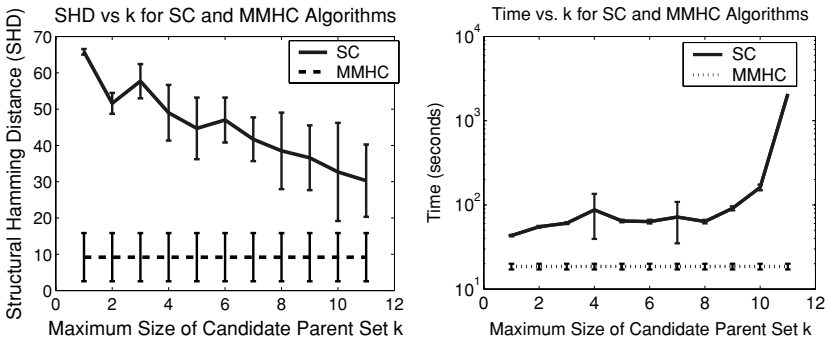


Fig. 14 Performance of the Sparse Candidate as k increases on reconstructing Alarm from 5000 cases

until there is no change in the candidate sets or a given number of iterations is performed with no improvement in the network score.

There are three main problems with Sparse Candidate. The first is that the estimation of the candidate sets is heuristic (may not identify the true set of parents), and it may take a number of iterations to converge to an approximation of the true set of parents. The second problem is that the user has to guess the parameter k, i.e., the maximum number of parents allowed for any node in the network. If the user overestimates k the algorithm will take unnecessarily long to finish and may even be rendered intractable for large datasets. If the user underestimates k there is a risk of discovering a suboptimal network. The final problem is the parameter k imposes a constraint of uniform sparseness on the network.

Figure 14 plots the average number of structural errors and running time over ten runs of Sparse Candidate on 5000-instance datasets sampled from ALARM as k increases. The *SHD* and running time of *MMHC* is also plotted as a baseline comparison (*MMHC* does not depend on k and is just plotted at all values of k for reference). As we can see, the number of errors highly depends on the value of k. Notice also that, Sparse Candidate exhibits a significant number of errors, even for relative large k values, despite the fact that all variables in ALARM have less than or equal to four parents. The cost of increasing the number of candidate parents is slower running times; running times which explode as k gets large. In fact, the Sparse Candidate's implementation we used will only run up to $k = 11$ on this problem.

Unfortunately, if a node has $m > k$ parents, then not only the node will be missing at least $m - k$ parents in the resulting network, but the errors may propagate to the rest of the network. A theoretical interpretation of this dependence follows.

Consider a node X with several parents, including Y (i.e., $Y \rightarrow X$) and let us assume Y failed to enter the candidate parents set of X, e.g., because X has more than k parents. Since Y is a parent of X, it has an association with X (if the network is faithful). That means that the hill-climbing search will erroneously tend to add the reverse edge $X \rightarrow Y$ to explain the association between the nodes. This in turn may prohibit other true edges from being added because they create cycles with the erroneously directed edge. In other words, we expect errors created by over-constraining the search (i.e., setting a low value for k) to propagate to other parts of the network.

Figure 15 shows a representative example. Sparse Candidate with the *Score* candidate parents selection method was run on 5000 instances sampled from the distribution of the ALARM network. Figure 15(a) shows a piece of the ALARM network structure around variable with

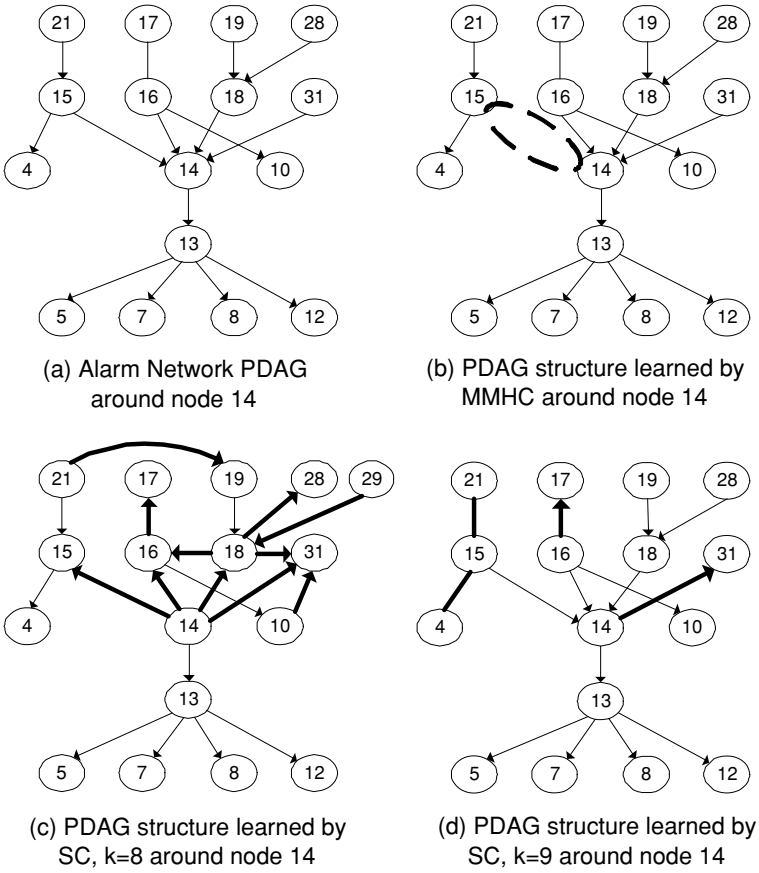


Fig. 15 The gold-standard network around node 14. Reconstructed area around node 14 of Alarm by *MMHC* (b) and by Sparse Candidate when $k = 8$ (c) and when $k = 9$ (d). Bold edges indicate the structural orientation errors made, dotted ovals indicate missing edges

index 14, which is the variable with the maximum in-degree of 4. The portion of the network shown is extracted from the PDAG representation of the network; the comparison here between structures is analogous to how the *SHD* metric is calculated. Parts (c) and (d) of the figure show the reconstructed networks (converted to PDAGs) learned by Sparse Candidate with $k = 8$ and $k = 9$ respectively after 3 iterations in each case. The bold edges denote structural errors. When $k = 8$, *not one but all four* of the parents {15, 16, 18, 31} of variable 14 are erroneously reported as children. The errors are propagated to other nodes, such as variable 19 and a number of other false positive edges are added to the structure. When k is increased to 9, the number of structural errors falls from 11 to 4, a sharp reduction. In contrast, for variable 14, the candidate parent set as estimated by *MMPC* is the set {15, 16, 18, 31, 13} that includes all the parents (and all children as it is supposed to). This allows hill-climbing to discover the correct structure (Fig. 15(b)) within one iteration with only one missing edge $15 \rightarrow 14$.

In general, Sparse Candidate uses simpler heuristics than *MMHC* to estimate the candidate sets, however, it requires iterating between search and re-estimation of the sets. *MMHC* on the other hand puts more effort in the initial selection of candidate sets so that only one search

cycle is adequate.[13] Our experiments indicate that the different strategy pays off not only in identifying better structures but also in significant computational savings.

13. Discussion, limitations, and future work

MMHC is a Bayesian network learning algorithm based on *MMPC*, an efficient and theoretically sound local learning algorithm. *MMPC*, and its extension *MMMB*, Max-Min Markov Blanket an algorithm for reconstructing Markov Blankets, were compared against other local learning algorithms, namely the Grow-Shrink (Margaritis & Thrun, 1999), the Koller-Sahami (Koller & Sahami, 1996), and the Incremental Association Markov Blanket (Tsamardinos, Aliferis & Statnikov, 2003b) and its variants. Compared to most of these algorithms, *MMPC* trades-off time for reduced sample size requirements (except the Koller-Sahami algorithm which is slower than *MMPC* and *MMMB*): Grow-Shrink and the Incremental Association Markov Blanket algorithms require sample size exponential the size of the local neighborhood learnt; in contrast, *MMPC* sample requirements depends on the connectivity of the structure learnt.

The local nature of *MMHC* has several advantages. First, each call to $\overline{MMPC}(T, \mathcal{D})$ considers edges only to and from T and thus, the skeleton identification phase naturally only requires one-dimensional tables to maintain the current list of edges considered. In contrast, some global methods begin with a two-dimensional connectivity matrix simultaneously considering all possible edges in the network. Once the skeleton has been identified, the graph of the network is stored as adjacency lists with very low memory requirements for sparse networks. This helps explain why *MMHC* is able to run on all networks in our experiments, while several other algorithms face memory problems on high-dimensional datasets.

A second advantage of the local approach method is that it can easily be modified to selectively reconstruct only a part of the network, when time constraints or increased complexity in a local area of the network forbids global reconstruction. In Tsamardinos et al. (2003a) we experimented with a variant of *MMHC* to reconstruct local regions of the network of an increasing radius edge-distance from a target node T. We intend to explore this research avenue further.

MMHC's heuristic heavily depends on measuring the strength of conditional association between a pair of variables, similar to the Three Phase Dependency Analysis algorithm. However, the latter uses the estimated mutual information to this end, instead of a statistical test, such as the χ^2. Both techniques use a cut-off threshold value, below or above which (depending on the metric) the strength of association is considered to be zero. Unfortunately, the estimated mutual information requires the use of a threshold that has no intuitive statistical meaning and *depends on the available sample relative to the conditioning set size* (degrees of freedom). For example, an estimated mutual information threshold value of 0.01 may be reasonable for the test $Ind(X; T \mid \{A, B, C\})$ when the available sample size is 1000. That is, let us assume that most likely, if X and T are indeed independent given $\{A, B, C\}$ their estimated mutual information from our sample will be lower than 0.01 and vice-versa. However, if the sample size drops to 100 cases, the 0.01 threshold may be too low since due to an increase in the variance of the estimation, the estimated mutual information is likely to be above 0.01 even when X and T are independent given $\{A, B, C\}$. The situation is similar when one fixes the available sample size but considers different conditioning

[13] In unreported experiments, we have allowed *MMHC* to continue iterating (re-estimate candidate parent sets using different methods), without any significant improvement in quality of reconstruction.

sets (or to be precise different degrees of freedom): an estimated mutual information value of 0.01 for the test $\mathbf{Ind}(X; T \mid \{A, B, C\})$ is not comparable to the same value for the test $\mathbf{Ind}(X; T \mid \{A, B, C, E\})$. In contrast the χ^2 test considers both the available sample and the degrees of freedom and the p values returned are comparable in all cases. In our experiments, the 5% threshold was used throughout for the χ^2 test with reliable and robust results.

Our experimental evaluation is extensive. Seven prototypical or state-of-the-art algorithms (thirteen different variations) were compared across twenty-two networks at three different samples sizes. Even so, some restrictions were made in terms of algorithms selected, scoring functions used by the algorithms, networks included in the study, sample sizes used, and the metrics used to quantify the evaluation results. We did not optimize the parameters of each algorithm for performance, or even better, compared the algorithms over a range of parameter values in a similar fashion to creating the Receiver Operating Curve.

In addition to the choice of algorithm, the implementation of each algorithm must also be considered. In Section 9.1.1, the implementation details of each algorithm are discussed. We mention that several possible optimizations suggested by Chickering (2002a) were not included in the version of the *GES* algorithm that was used in this study. In the future, we would like to replace this version of the *GES* algorithm by one by the original authors.

In this study, when a scoring function is used to quantify the network quality the BDeu scoring function is chosen with an equivalent sample size of 10. This choice has become fairly standard practice taken by many algorithm implementations in the literature (Heckerman, Geiger & Chickering, 1995; Friedman, Nachman and Pe'er, 1999; Chickering, 2002b). In Acid et al. (2004), several scoring functions are compared. Future work could include a similar study of the effects of scoring function for each algorithm.

The networks that were chosen for this study were mainly taken from real decision support systems. The hope is that these networks well represent the kind of distributions most frequently encountered in practice. The networks selected are all fairly sparse. Additional networks that are more dense (whether taken from the real-world or manually constructed) could be considered in future work.

The sample sizes analyzed is limited: 500, 1000, 5000 and a smaller set of tests at 20,000. An interesting study would be to analyze the behavior of the algorithms with extremely small sample (100 or less) and explore adjustments of the algorithms for very small sample. Data sets with very limited sample are abundant, especially new data sets coming from biology with the advent of wide-spread use of genomic and proteomic data.

MMHC is a promising new algorithm that outperformed all others included in our experiments. Nevertheless, the algorithms still has several limitations to consider.

The first limitation is that the algorithm requires a network to be faithful, or close to faithful, to reconstruct. The faithfulness assumption can be relaxed from what stated: *MMPC*'s proof of correctness would still be valid if we require a limited and local form of faithfulness, where a network is locally-faithful if for every pair of nodes connected by an edge, the nodes are associated conditioned on any other subset of variables. However, even with this relaxation the algorithm will still not be able to identify the parents of a variable connected to it via a parity or similar function. In the future, we intend to work on relaxing this assumption.

Other possible extensions to the algorithm are to incorporate statistical tests targeting specific distributions, employing parametric assumptions, or incorporate background knowledge.

For example, parametric assumptions could lead to more statistically powerful tests (requiring less sample size) and allow a power analysis of the tests of independence.

In preliminary work we have evaluated a polynomial version *MMPC* with promising results (Brown, Tsamardinos & Aliferis, 2005). The resulting version of *MMHC* using polynomial MMPC scales up better with increased sample and is more efficient while it maintains the same quality of learning. We intend to explore further polynomial extensions of *MMHC* and identify conditions of correctness similar to monotone faithfulness (Cheng, Bell & Liu, 1998).

Other interesting research avenues to explore are extensions and modifications to the greedy search-and-score procedure used for edge orientation. One could couple the sound edge identification constraint-based method we use with a sound search and score technique, such as the Greedy Equivalence Search to obtain a sound algorithm in the sample limit. We also note and remind, that in our experiment with a 5000-variable tiled-ALARM network (Section 9.2.7) the skeleton reconstruction phase took about 19 hours, while the search-and-score part required another 12 days time approximately. This indicates, that the most promising part to optimize, in terms of computational gains, is the edge orientation phase.

14. Conclusion

In this paper we presented a new algorithm for Bayesian network structure learning, called Max-Min Hill-Climbing (*MMHC*). The algorithm combines ideas from local learning, constraint-based, and search-and-score techniques in a principled and effective way. It first reconstructs the skeleton of a Bayesian network and then performs a Bayesian-scoring greedy hill-climbing search to orient the edges. *MMHC* pushes the envelope of Bayesian network learning to datasets with thousands of variables while simultaneously improving the reconstruction quality over existing algorithms. In our experiments, on average, *MMHC* outperformed both in terms of time efficiency (subject to the implementations selected) and quality of reconstruction the *PC*, the Sparse Candidate, the Three Phase Dependency Analysis, the Optimal Reinsertion, the Greedy Equivalence Search, and the Greedy Search algorithms on a variety of networks, sample sizes, and parameter values.

As a constraint-based algorithm, *MMHC* improves the search strategy for conditional independencies of the *PC* algorithm by employing a more powerful heuristic and by lifting the restriction to perform constraint-based tests in order of increasing conditioning set size. In addition, it corrects several problems of *PC* when the sample is low, indicating that *PC*'s counter-intuitive behavior in low samples is not inherent to constraint-based methods.

As a hybrid algorithm, *MMHC* improves on the ideas by Sparse Candidate by employing a sound method to constrain a subsequent search-and-score edge orientation phase. As a result, the user is not required to estimate the parameter k of the maximum number of parents allowed for a node and the network does not have to be uniformly sparse to reconstruct. We show that underestimating the parameter k in Sparse Candidate may lead to errors propagating to other parts of the reconstructed network, while overestimating k leads to increased computational efforts spend. Finally, unlike the Sparse Candidate, *MMHC* only requires one iteration of candidate parent estimation and search to learn a network, leading to significant computational savings.

Our extensive experimental evaluation is one of the first simultaneously comparing most of the major Bayesian network algorithms against each other. The results support our theoretical analysis stated above regarding *PC* and Sparse Candidate. The detailed results, datasets, algorithms, experimentation scripts, code to compute the performance metrics used, and

other useful information are available on-line to facilitate researchers to repeat the experiments with new algorithms and/or implementations (http://www.dsl-lab.org/supplements/mmhc_paper/mmhc_index.html). The version of *MMHC* used in our experiments is available at the same URL as above, and as part of the Causal Explorer 1.3 and its future versions.

Appendix A: Proof of correctness

Lemma 1. $\mathbf{PC}_T \subseteq \overline{MMPC}(T, \mathcal{D})$.

Proof: If $X \in \mathbf{PC}_T$ then because of faithfulness (Theorem 2), $Dep(X; T|\mathbf{Z})$ and $Assoc(X; T|\mathbf{Z}) > 0$, for any $\mathbf{Z}$. Thus, X will be selected by the Max-Min Heuristic in some iteration before the algorithm moves to the Phase II and it will eventually enter **CPC**; once it enters it can never be removed because the independence tests will always fail for X (Algorithm 1 Line 10). □

Lemma 2. *If $X \in \overline{MMPC}(T; \mathcal{D})$ and $X \notin \mathbf{PC}_T$, then X is a descendant of T in all networks G faithful to the distribution of the data.*

Proof: Let G be the graph of any Bayesian network that faithfully captures the distribution of the data $\mathcal{D}$ (there has to be at least one since we assume a faithful distribution). By Lemma 1 the following holds: $\mathbf{Pa}_T^G \subseteq \mathbf{PC}_T \subseteq \overline{MMPC}(\mathbf{T}; \mathbf{D})$.

In Phase II of $\overline{MMPC}$ we condition on at least all subsets of the output $\overline{MMPC}(T; \mathcal{D})$, and so at some point we will condition on $\mathbf{Pa}_T^G$. By the Markov Condition in G we know that $Ind(X; T \mid \mathbf{Pa}_T^G)$ for any non-descendant variable $X \in \mathcal{V} \setminus \mathbf{Pa}_T^G$. Thus, if $X \in \overline{MMPC}(T; \mathcal{D})$ then either $X \in \mathbf{Pa}_T^G$ or X is a descendant of T in G. The latter condition has to hold since we assumed $X \notin \mathbf{PC}_T$. □

Theorem 3. *Algorithm $MMPC(T, \mathcal{D})$ will return $\mathbf{PC}_T$.*

Proof: If $X \in \mathbf{PC}_T$, then $T \in \mathbf{PC}_X$ and by Lemma 1, $X \in \overline{MMPC}(T; \mathcal{D})$ and $T \in \overline{MMPC}(X; \mathcal{D})$. Thus, X will be included in **CPC** at Line 2 of *MMPC* (Algorithm 2, and the condition at Line 4 will fail; in the end X will be included in $MMPC(T; \mathcal{D})$. Thus, *MMPC* returns all members of $\mathbf{PC}_T$.

Let us now suppose that X is returned by $MMPC(T, \mathcal{D})$ and that $X \notin \mathbf{PC}_T$ (and so $T \notin \mathbf{PC}_X$). Since X passed the test at Line 4 (Algorithm 2), $X \in \overline{MMPC}(T, \mathcal{D})$ and $T \in \overline{MMPC}(X, \mathcal{D})$.

Thus, by Lemma 2 X is a descendent of T in any G faithful to the distribution of the data. By the same Lemma T is also a descendent of X in T. Since, Bayesian networks are acyclic this is a contradiction and so X cannot be returned by $MMPC(T, \mathcal{D})$ unless $X \in \mathbf{PC}_T$. Thus, *MMPC* returns only members of $\mathbf{PC}_T$. □

Acknowledgments The first and third author are supported by the National Library of Medicine grant LM-7948-01. The second author is supported by the National Library of Medicine grant T15 LM07450-01. We are in debt to Alexander Statnikov for parts of the code, testing, and useful suggestions and to Yerbolat Dosbayev for help with some implementation issues. We would like to thank Sonia M. Leach and Douglas H. Fisher for their useful feedback. We would like to thank the authors of the Sparse Candidate, Tetrad, and Optimal Reinsertion for making publicly available their algorithms and systems for inclusion in this study. Finally, we would like to thank the editor and the anonymous reviewers for their helpful feedback and for providing a draft of the computational complexity comparison between the Greedy Search and *MMHC*.

References

Abramson, B., Brown, J., Edwards, W., Murphy, A., & Winkler, R. L. (1996). Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, *12*, 57–71.

Acid, S., de Campos, L., Fernandez-Luna, J., Rodriguez, S., Rodriguez, J., & Salcedo, J. (2004). A comparison of learning algorithms for Bayesian networks: A case study based on data from an emergency medical service. *Artificial Intelligence in Medicine, 30*, 215–232.

Acid, S., & de Campos, L. M. (2003). Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs. *Journal of Artificial Intelligence Research*, 445–490.

Acid, S., & de Campos, L. (2001). A hybrid methodology for learning belief networks: BENEDICT. *International Journal of Approximate Reasoning*, 235–262.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*, 716–723.

Aliferis, C. F., Tsamardinos, I., Statnikov, A., & Brown, L. E. (2003a). Causal explorer: A causal probabilistic network learning toolkit for biomedical discovery. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS '03)* (pp. 371–376).

Aliferis, C. F., Tsamardinos, I., & Statnikov, A. (2003b). HITON, A novel markov blanket algorithm for optimal variable selection. In *American Medical Informatics Association (AMIA)* (pp. 21–25).

Andreassen, S., Jensen, F. V., Andersen, S. K., Falck, B., Kharulff, U., & Woldbye, M. (1989). MUNIN—An expert EMG assistant. In J. E. Desmedt (Eds.), *Computer-aided electromyography and expert systems*.

Baeze-Yates, R., & Ribiero-Neto, B. (1999). *Modern information retrieval*. Addison-Wesley Pub Co.

Beal, M. J. & Ghahramani, Z. (2003). The variational Bayesian EM algorithm for incomplete data: With application to scoring graphical model structures. In J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith, & M. West (Eds.), *Bayesian statistics 7*. Oxford University Press.

Beinlich, I. A., Suermondt, H., Chavez, R., Cooper, G., et al. (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Second European Conference in Artificial Intelligence in Medicine*.

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

Bouckaert, R. (1995). Bayesian belief networks from construction to inference. Ph.D. thesis, University of Utrecht.

Brown, L., Tsamardinos, I., & Aliferis, C. (2004). A novel algorithm for scalable and accurate bayesian network learning. In *11th World Congress on Medical Informatics (MEDINFO)*. San Francisco, California.

Brown, L. E., Tsamardinos, I., & Aliferis, C. F. (2005). A comparison of novel and state-of-the-art polynomial Bayesian network learning algorithms. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*.

Chapman, W. W., Fizman, M., Chapman, B. E. & Haug, P. J. (2001). A comparison of classification algorithms to automatically identify chest X-ray reports that support pneumonia. *Journal of Biomedical Informatics, 34*, 4–14.

Cheng, J., Bell, D., & Liu, W. (1998). Learning Bayesian networks from data: An efficient approach based on information theory. Technical report, University of Alberta, Canada.

Cheng, J., Greiner, R., Kelly, J., Bell, D. A. & Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence, 137*, 43–90.

Chickering, D. (1995). A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*. San Francisco, CA (pp. 87–98). Morgan Kaufmann Publishers.

Chickering, D. (1996). Learning Bayesian networks is NP-complete. In D. Fisher and H. Lenz (Eds.), *Learning from data: Artificial intelligence and statistics V* (pp. 121–130) Springer-Verlag.

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

Chickering, D., Geiger, D. & Heckerman, D. (1995). Learning Bayesian networks: Search methods and experimental results. In *Fifth International Workshop on Artificial Intelligence and Statistics* (pp. 112–128).

Chickering, D., Meek, C. & Heckerman D. (2004). Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research, 5*, 1287–1330.

Chickering, D. M. (2002a). Optimal structure identification with greedy search. *Journal of Machine Learning Research,* 507–554.

Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning, 9*(4), 309–347.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., & Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems*. Springer.

Dash, D. (2005). Restructuring dynamic causal systems in equilibrium. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AIStats 2005)*.

Dash, D. & Druzdzel, M. (1999). A hybrid anytime algorithm for the construction of causal models from sparse data. In *Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*.

Dash, D., & Druzdzel, M. (2003). Robust independence testing for constraint-based learning of causal structure. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)* (pp. 167–174), Morgan Kaufmann.

Dor, D., & Tarsi, M. (1992). A simple algorithm to construct a consistent extension of a partially oriented graph. Technicial Report R-185, Cognitive Systems Laboratory, UCLA.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*. (pp. 129–138), San Francisco, CA, Morgan Kaufmann Publishers.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *Computational Biology, 7*, 601–620.

Friedman, N., Nachman, I., & Pe'er, D., (1999). Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*.

Ghahramani, Z., & Beal, M. (2001). Graphical models and variational methods. In M. Opper, & D. Saad (Eds.), *Advanced mean field methods—Theory and practice*. MIT Press.

Glymour, C., & Cooper, G. F. (eds.) (1999). *Computation, causation, and discovery*. AAAI Press/The MIT Press.

Glymour, C. N. (2001). *The mind's arrows: Bayes nets & graphical causal models in psychology*. MIT Press.

Goldenberg, A., & Moore, A. (2004). Tractable learning of large Bayes net structures from sparse data. In *Proceedings of 21st International Conference on Machine Learning*.

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

Jensen, A., & Jensen, F. (1996). Midas—An influence diagram for management of mildew in winter wheat. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)* (pp. 349–356). Morgan Kaufmann Publishers.

Jensen, C. S. (1997). Blocking Gibbs sampling for inference in large and complex Bayesian networks with applications in genetics. Ph.D. thesis, Aalborg University, Denmark.

Jensen, C. S., & Kong, A. (1996). Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops. Research Report R-96-2048, Department of Computer Science, Aalborg University, Denmark.

Jordan, M. I., Ghahramani, Z., T.S., J., & L.K., S. (1999). An introduction to variational methods for graphical models. *Machine Learning, 37*, 183–233.

Kocka, T., Bouckaert, R., & Studeny, M. (2001). On the inclusion problem. Technical report, Academy of Sciences of the Czech Republic.

Kovisto, M., & Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research, 5*, 549–573.

Koller, D., & Sahami, M. (1996). Toward optimal feature selection. In *Thirteen International Conference in Machine Learning*.

Komarek, P., & Moore, A. (2000). A dynamic adaptation of AD-trees for efficient machine learning on large data sets. In *Proc. 17th International Conf. on Machine Learning* (pp. 495–502). San Francisco, CA: Morgan Kaufmann.

Kristensen, K., & Rasmussen, I. A. (2002). The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture, 33*, 197–217.

Kullback, S., & Leibler, R. (1951). On information and sufficiency. *Annals of Mathematical Statistics, 22*, 79–86.

Margaritis, D., & Thrun, S. (1999). Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12 (NIPS)*.

Margaritis, D., & Thrun, S. (2001). A Bayesian multiresolution independence test for continuous variables. In *17th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Meek, C. (1995). Strong completeness and faithfulnes in Bayesian networks. In *Conference on Uncertainty in Artificial Intelligence* 411–418.

Meek, C. (1997). Graphical models: Selecting causal and statistical models. Ph.D. thesis, Carnegie Mellon University.

Moore, A., & Lee, M. (1998). Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research, 8*, 67–91.

Moore, A., & Schneider, J. (2002). Real-valued all-dimensions search: Low-overhead rapid searching over subsets of attributes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-2002)* (pp. 360–369).

Moore, A., & Wong, W. (2003). Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *Twentieth International Conference on Machine Learning (ICML-2003)*.

Neapolitan, R. (2003). *Learning Bayesian networks*. Prentice Hall.

Nielson, J., Kocka, T., & Pena, J. (2003). On local optima in learning bayesian networks. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence,* 435–442.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. San Mateo, CA: Morgan Kaufmann.

Pearl, J. (2000). *Causality, models, reasoning, and inference*. Cambridge University Press.

Pearl, J., & Verma, T. (1991). A theory of inferred causation. In J. F. Allen, R. Fikes, & E. Sandewall (Eds.), *KR'91: Principles of knowledge representation and reasoning* (pp. 441–452). San Mateo, California: Morgan Kaufmann.

Peterson, W., TG, B., & Fox, W. (1954). The theory of signal detectability. *IRE Professional Group on Information Theory PGIT-4,* 171–212.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica, 14*, 465–671.

Rissanen, J. (1987). Stochastic complexity. *Journal of the Royal Statistical Soceity, Series B, 49*, 223–239.

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

Silverstein, C., Brin, S., Motwani, R., & Ullman, J. (2000). Scalable techniques for mining causal structures. *Data Mining and Knowledge Discovery, 4*(2/3), 163–192.

Singh, M., & Valtorta, M. (1993). An algorithm for the construction of Bayesian network structures from data. In *9th Conference on Uncertainty in Artificial Intelligence,* pp. 259–265.

Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K. *et al.* & Eisen, M. B. (1998). Comprehensive identification of cell cycle regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell, 9*, 3273–3297.

Spirtes, P., Glymour, C., & Scheines, R. (1990). Causality from probability. In J. Tiles, G. McKee, & G. Dean (eds.): *Evolving knowledge in the natural and behavioral sciences* (pp. 181–199). London: Pittman.

Spirtes, P., Glymour, C. & Scheines, R. (1993). *Causation, prediction, and search*. Springer/Verlag, first edition.

Spirtes, P., Glymour, C., & Scheines, R. (2000). *Causation, prediction, and search*. The MIT Press, second edition.

Spirtes, P., & Meek, C. (1995). Learning Bayesian networks with discrete variables from data. In *Proceedings from First Annual Conference on Knowledge Discovery and Data Mining* (pp. 294–299). Morgan Kaufmann.

Statnikov, A., Tsamardinos, I., & Aliferis, C. F. (2003). An algorithm for the generation of large Bayesian networks. Technical Report DSL-03-01, Vanderbilt University.

Steck, H., & Jaakkola, T. (2002). On the dirichlet prior and Bayesian regularization. In *Advances in Neural Information Processing Systems, 15*.

Tsamardinos, I., & Aliferis, C. F. (2003). Towards principled feature selection: Relevancy, filters and wrappers. In *Ninth International Workshop on Artificial Intelligence and Statistics (AI & Stats 2003)*.

Tsamardinos, I., Aliferis, C. F., & Statnikov, A. (2003b). Algorithms for large scale markov blanket discovery. In *The 16th International FLAIRS Conference* (pp. 376–381).

Tsamardinos, I., Aliferis, C. F., & Statnikov, A. (2003c). Time and sample efficient discovery of Markov blankets and direct causal relations. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 673–678).

Tsamardinos, I., Aliferis, C. F., & Statnikov, A. (2003a). Time and sample efficient discovery of Markov Blankets and direct causal relations. Technical Report DSL-03-02, Vanderbilt University.

Tsamardinos, I., Aliferis, C. F., Statnikov, A., & Brown. L. E. (2003a). Scaling-Up Bayesian network learning to thousands of variables using local Learning Technique. Technical Report DSL TR-03-02, Dept. Biomedical Informatics, Vanderbilt University.

Tsamardinos, I., Statnikov, A., Brown, L. E., and Aliferis, C. F. (2006) Generating realistic large bayesian networks by tiling. In *The 19th International FLAIRS Conference* (to appear).

Verma, T., & Pearl, J. (1988). Causal networks: Semantics and expressiveness. In: *4th Workshop on Uncertainty in Artificial Intelligence*.

Verma, T., & Pearl, J. (1990). Equivalence and synthesis of causal models. In *Proceedins of 6th Annual Conference on Uncertainty in Artificial Intelligence* (pp. 255–268). Elsevier Science.