Journal of Machine Learning Research 9 (2008) 2251-2286 Submitted 12/07; Revised 6/08; Published 10/08

Finding Optimal Bayesian Network Given a Super-Structure

Eric Perrier PERRIER@IMS.U-TOKYO.AC.JP
Seiya Imoto IMOTO@IMS.U-TOKYO.AC.JP
Satoru Miyano MIYANO@IMS.U-TOKYO.AC.JP
Human Genome Center, Institute of Medical Science
University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan

Editor: Max Chickering

Abstract

Classical approaches used to learn Bayesian network structure from data have disadvantages in terms of complexity and lower accuracy of their results. However, a recent empirical study has shown that a hybrid algorithm improves sensitively accuracy and speed: it learns a skeleton with an independency test (IT) approach and constrains on the directed acyclic graphs (DAG) considered during the search-and-score phase. Subsequently, we theorize the structural constraint by introducing the concept of super-structure S, which is an undirected graph that restricts the search to networks whose skeleton is a subgraph of S. We develop a super-structure constrained optimal search (COS): its time complexity is upper bounded by $O(\gamma_m{}^n)$, where $\gamma_m < 2$ depends on the maximal degree m of S. Empirically, complexity depends on the average degree $\tilde{m}$ and sparse structures allow larger graphs to be calculated. Our algorithm is faster than an optimal search by several orders and even finds more accurate results when given a sound super-structure. Practically, S can be approximated by IT approaches; significance level of the tests controls its sparseness, enabling to control the trade-off between speed and accuracy. For incomplete super-structures, a greedily post-processed version (COS+) still enables to significantly outperform other heuristic searches.

Keywords: Bayesian networks, structure learning, optimal search, super-structure, connected subset

1. Introduction

It is impossible to understand large raw sets of data obtained from a huge number of correlated variables. Therefore, in order to simplify the comprehension of the system, various graphical models have been developed to summarize interactions between such variables in a synoptic graph. Among the existing models, Bayesian networks have been widely employed for decades in various domains including artificial intelligence (Glymour, 2001), medicine (Cowell et al., 1999), bioinformatics (Friedman et al., 2000), and even economy (Segal et al., 2005) and sociology (Heckerman, 1996). Bayesian networks compactly represent a joint probability distribution P over the set of variables, using DAG to encode conditional independencies between them (Pearl, 1988). The popularity of this model is primarily due to its high expressive power, enabling the simultaneous investigation of complex relationships between many variables of a heterogeneous nature (discrete or continuous). Further, for Bayesian network model inference from data is comparatively simpler; incomplete or noisy data are also usable and prior knowledge can be incorporated. When the DAG or structure of

the model is known, the parameters of the conditional probability distributions can be easily fit to the data; thus, the bottleneck of modeling an unknown system is to infer its structure.

Over the previous decades, various research directions have been explored through a numerous literature to deal with structure learning, which let us propose the following observations. Maximizing a score function over the space of DAGs is a promising approach towards learning structure from data. A search strategy called optimal search (OS) have been developed to find the graphs having the highest score (or global optima) in exponential time. However, since it is feasible only for small networks (containing up to thirty nodes), in practice heuristic searches are used. The resulting graphs are local optima and their accuracy strongly depends on the heuristic search strategy. In general, given no prior knowledge, the best strategy is still a basic greedy hill climbing search (HC). In addition, Tsamardinos et al. (2006) proposed to constrain the search space by learning a skeleton using an IT-based technique before proceeding to a restricted search. By combining this method with a HC search, they developed a hybrid algorithm called max-min hill-climbing (MMHC) that is faster and usually more accurate.

In the present study we are interested in OS since the optimal graphs will converge to the true model in the sample limit. We aim to improve the speed of OS in order to apply it to larger networks; for this, a structural constraint could be of a valuable help. In order to keep the asymptotic correctness of OS, the constraint has to authorize at least the edges of the true network, but it can contain also extra edges. Following this minimal condition that should respect a constraint on the skeletons to be sound, we formalize a flexible structural constraint over DAGs by defining the concept of a super-structure. This is an undirected graph that is assumed to contain the skeleton of the true graph (i.e., the true skeleton). In other word, the search space is the set of DAGs that have a subgraph of the given super-structure as a skeleton. A sound super-structure (that effectively contains the true skeleton) could be provided by prior knowledge or learned from data much more easily (with a higher probability) than the true skeleton itself. Subsequently, we consider the problem of maximizing a score function given a super-structure, and we derive a constrained optimal search, COS, that finds a global optimum over the restricted search space. Not surprisingly, our algorithm is faster than OS since the search space is smaller; more precisely, its computational complexity is proportional to the number of connected subsets of the super-structure. An upper bound is derived theoretically and average complexity is experimentally showed to depend on the average degree of the super-structure. Concretely, for sparse structures our algorithm can be applied to larger networks than OS (with an average degree around 2.1, graphs having 1.6 times more nodes could be considered). Moreover, for a sound super-structure, learned graphs are more accurate than unconstrained optima: this is because, some incorrect edges are forbidden, even if their addition to the graph improves the score.

Since the sparseness directly affects the speed, and therefore the feasibility of our search, it remains to propose efficient methods to learn a sound and sparse super-structures without prior knowledge. This is out of the scope of this present paper where we focus on the enunciation of our constraint, its application to optimal search and optimizations of its implementation. Nevertheless, in order to demonstrate our algorithm in practice, we propose a first basic strategy to approximate a super-structure from data. The idea is to use "relaxed" independency testing to obtain an undirected graph that may contain the true skeleton with a high probability, while yet being sparse. In that case, we can consider the significance level of the independency tests, α, as a tool to choose between accuracy (high values return dense but probably sound structures) and speed (low values give sparse but incomplete structures). We tested our proposition on MMPC, the IT-based strategy

used by Tsamardinos et al. (2006) in MMHC; our choice was motivated by the good results of their algorithm that we also include in our comparative study. MMPC appears to be a good method to learn robust and relatively sparse skeletons; unfortunately, soundness is achieved only for high significance levels, $\alpha > 0.9$, implying a long calculation and a denser structure. Practically, when the constraint is learned with $\alpha = 0.05$, in terms of accuracy, COS is worse than OS since the super-structure is usually incomplete; still, COS outperforms most of the time greedy searches, although it finds graphs of lower scores. Resulting graphs can be quickly improved by applying to them a post-processing unconstrained hill-climbing (COS+). During that final phase, scores are strictly improved, and usually accuracy also. Interestingly, even for really low significance levels ($\alpha \approx 10^{-5}$), COS+ returns graphs more accurate and of a higher score than both MMHC and HC. COS+ can be seen as a bridge between HC (when α tends to 0) and OS (when α tends to 1) and can be applied up to a hundred nodes by selecting a low enough significance level.

This paper is organized as follows. In Section 2, we discuss the existing literature on structure learning. We clarify our notation in Section 3.1 and reintroduce OS in Section 3.2. Then, in Section 4, the core of this paper, we define super-structures and present our algorithm, proofs of its complexity and practical information for implementation. Section 5 details our experimental procedures and presents the results. Section 5.1.4 briefly recalls MMPC, the method we used during experiments to learn the super-structures from data. Finally, in Section 6, we conclude and outline our future works.

2. Related Works

The algorithms for learning the Bayesian network structure that have been proposed until now can be regrouped into two different approaches, which are described below.

2.1 IT Approach

This approach includes IC algorithm (inductive causation) (Pearl, 1988), PC algorithm (after its authors, Peter and Clark) (Spirtes et al., 2000), GS algorithm (grow and shrink) (Margaritis and Thrun, 2000), and TPDA algorithm (three-phase dependency analysis) (Cheng et al., 2002). All of them build the structure to be consistent with the conditional independencies among the variables that are evaluated with a statistical test (G-square, partial correlation). Usually, algorithms start by learning the skeleton of the graph (by propagating constraints on the neighborhood of each variable) and then edges are oriented to cope with dependencies revealed from data. Finally, one network is retained from the equivalent class consistent with the series of tests. Under the faithful condition of *P*, such strategies have been proven to build a graph converging to the true network as the size of the data approaches infinity. Moreover, their complexity is polynomial, assuming that the maximal degree of the network, that is, the maximal size of nodes neighborhood, is bounded (Kalisch and Bühlmann, 2007). However, in practice, the results are mixed because of the tests sensitivity to noise: since these algorithms base their decisions on a single or few tests, they are prone to accumulate errors (Margaritis and Thrun, 2000). Worse, they can obtain a set of conditional independencies that is contradictory, or that cannot be faithfully encoded by a DAG, leading to a failure of the algorithm. Moreover, except for sparse graphs, their execution time is generally longer than that of algorithms from the scoring criteria-based approach (Tsamardinos et al., 2006).

2.2 Scoring Criteria-Based Approach

Search-and-score methods are favored in practice and considered as a more promising research direction. This second family of algorithms uses a scoring criterion, such as the posterior probability of the network given the data, in order to evaluate how well a given DAG fits empirical results, and returns the one that maximized the scoring function during the search. Since the search space is known to be of a super exponential size on the number of nodes n, that is, $O(n!2^{\binom{n}{2}})$ (Robinson, 1973), an exhaustive search is practically infeasible, implying that various greedy strategies have been proposed to browse DAG space, sometimes requiring some prior knowledge.

Among them, the state-of-the-art greedy hill climbing (HC) strategy, although it is simple and will find only a locally optimal network, remains one of the most employed method in practice, especially with larger networks. There exist various implementations using different empirical tricks to improve the score of the results, such as TABU list, restarting, simulated annealing, or searching with different orderings of the variables (Chickering et al., 1995; Bouckaert, 1995). However a traditional and basic algorithm will process in the following manner:

- Start the search from a given DAG, usually the empty one.
- Then, from a list of possible transformations containing at least addition, withdrawal or reversal of an edge, select and apply the transformation that improves the score most while also ensuring that graph remains acyclic.
- Finally repeat previous step until strict improvements to the score can no longer be found.

More details about our implementation of HC are given in Section 5.1.3. Such an algorithm can be used even for large systems, and if the number of variables is really high, it can be adapted by reducing the set of transformations considered, or by learning parents of each node successively. In any case, this algorithm finds a local optimum DAG but without any assertion about its accuracy (besides its score). Further, the result is probably far from a global optimal structure, especially when number of nodes increases. However, optimized forms of this algorithm obtained by using one or more tricks have been considered to be the best search strategies in practice until recently.

Other greedy strategies have also been developed in order to improve either the speed or accuracy of HC one: sparse candidate (SC, Friedman et al., 1999) that limits the maximal number of parents and estimate candidate parents for each node before the search, greedy equivalent search (GES, Chickering, 2002b) that searches into the space of equivalence classes (PDAGs), and optimal reinsertion (OR, Moore and Wong, 2003) that greedily applies an optimal reinsertion transformation repeatedly on the graph.

SC was one of the first to propose a reduction in the search space, thereby sensitively improving the score of resulting networks without increasing the complexity too much if candidate parents are correctly selected. However, it has the disadvantage of a lack of flexibility, since imposing a constant number of candidate parents to every node could be excessive or restrictive. Furthermore, the methods and measures proposed to select the candidates, despite their intuitive interest, have not been proved to include at least the true or optimal parents for each node.

GES has the benefit that it exploits a theoretically justified direction. Main scoring functions have been proved to be score equivalent (Chickering, 1995), that is, two equivalent DAGs (representing the same set of independencies among variables) have the same score. Thus they define

equivalent classes over networks that can be uniquely represented by CPDAGs. Therefore, searching into the space of equivalent classes reduces the number of cases that have to be considered, since one CPDAG represents several DAGs. Further, by using usual sets of transformations adapted to CPDAGs, the space browsed during a greedy search becomes more connected, increasing the chances of finding a better local maximum. Unfortunately, the space of equivalent classes seems to be of the same size order than that of DAGs, and an evaluation of the transformations for CPDAGs is more time consuming. Thus, GES is several times slower than HC, and it returns similar results. Interestingly, following the comparative study of Tsamardinos et al. (2006), if structural errors rather than scores are considered as a measure of the quality of the results, GES is better than a basic HC.

In the case of OR, the algorithm had the advantage to consider a new transformation that globally affects the graph structure at each step: this somehow enables the search to escape readily from local optima. Moreover, the authors developed efficient data-structures to rationalize score evaluations and retrieve easily evaluation of their operators. Thus, it is one of the best greedy methods proposed; however, with increasing data, the algorithm will collapse due to memory shortage.

Another proposed direction was using the K2 algorithm (Cooper and Herskovits, 1992), which constraints the causal ordering of variables. Such ordering can be seen to be a topological ordering of the true graph, provided that such a graph is acyclic. Based on this, the authors proposed a strategy to find an optimal graph by selecting the best parent set of a node among the subsets of nodes preceding it. The resulting graph can be the global optimal DAG if it accepts the same topological ordering. Therefore, given an optimal ordering, K2 can be seen as an optimal algorithm with a time and space complexity of $O(2^n)$. Moreover, for some scoring functions, branch-pruning can be used while looking for the best parent set of a node (Suzuki, 1998), thereby improving the complexity. However, in practice, a greedy search that considers adding and withdrawing a parent is applied to select a locally optimal parent set. In addition, the results are strongly depending on the quality of the ordering. Some investigations have been made to select better orderings (Teyssier and Koller, 2005) with promising results.

2.3 Recent Progress

One can wonder about the feasibility of finding a globally optimal graph without having to explicitly check every possible graph, since nothing can be asserted with respect to the structural accuracy of the local maxima found by previous algorithms. In a general case, learning Bayesian network from data is an NP-hard problem (Chickering, 1996), and thus for large networks, only such greedy algorithms are used. However, recently, algorithms for global optimization or exact Bayesian inference have been proposed (Ott et al., 2004; Koivisto and Sood, 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006) and can be applied up to a few tens of nodes. Since they all principally share the same strategy that we will introduce in detail subsequently, we will refer to it as optimal search (OS). Even if such a method cannot be of a great use in practice, it could validate empirically the search-and-score approach by letting us study how a global maximum converges to the true graph when the data size increases; Also, it could be a meaningful gold standard to judge the performances of greedy algorithms.

Finally, a recent noteworthy step was performed with the min-max hill climbing algorithm (MMHC, Tsamardinos et al., 2006), since it was empirically proved to be the fastest and the best method in terms of structural error based on the structural hamming distance. This algorithm can be considered as a hybrid of the two approaches. It first learns an approximation of the skeleton of

the true graph by using an IT strategy. It is based on a subroutine called min-max-parents-children (MMPC) that reconstructs the neighborhood of each node; G-square tests are used to evaluate conditional independencies. The algorithm subsequently proceeds to a HC search to build a DAG limiting edge additions to the one present in the retrieved skeleton. As a result, it follows a similar technique than that of SC, except that the number of candidate parents is tuned adaptively for each node, and that the chosen candidates are sound in the sample limit. It is worth to notice that the skeleton learned in the first phase can differ from the one of the final DAG, since all edges will not be for sure added during the greedy search. However, it will be certainly a cover of the resulting graph skeleton.

3. Definitions and Preliminaries

In this section, after explaining our notations and recalling some important definitions and results, we discuss structure constraining and define the concept of a super-structure. Section 3.3 is dedicated to OS.

3.1 Notation and Results for Bayesian Networks

In the rest of the paper, we will use upper-case letters to denote random variables (e.g., X_i, V_i) and lower-case letters for the state or value of the corresponding variables (e.g., x_i, v_i). Bold-face will be used for sets of variables (e.g., $\mathbf{Pa}_i$) or values (e.g., $\mathbf{pa}_i$). We will deal only with discrete probability distributions and complete data sets for simplicity, although a continuous distribution case could also be considered using our method.

Given a set $\mathbf{X}$ of n random variables, we would like to study their probability distribution P_0. To model this system, we will use Bayesian networks:

Definition 1. (Pearl, 1988; Spirtes et al., 2000; Neapolitan, 2003) *Let P be a discrete joint probability distribution of the random variables in some set* $\mathbf{V}$*, and* $G = (\mathbf{V}, \mathbf{E})$ *be a directed acyclic graph (DAG). We call* (G, P) *a* Bayesian network (BN) *if it satisfies the Markov condition, that is, each variable is independent of any subset of its non-descendant variables conditioned on its parents.*

We will denote the set of the parents of a variable V_i in a graph G by $\mathbf{Pa}_i$, and by using the Markov condition, we can prove that for any BN (G, P), the distribution P can be factored as follows:

$$P(\mathbf{V}) = P(V_1, \cdots, V_p) = \prod_{V_i \in \mathbf{V}} P(V_i | \mathbf{Pa}_i).$$

Therefore, to represent a BN, the graph G and the joint probability distribution have to be encoded; for the latter, every probability $P(V_i = v_i | \mathbf{Pa}_i = \mathbf{pa}_i)$ should be specified. G directly encodes some of the independencies of P and entails others (Neapolitan, 2003). More precisely, all independencies entailed in a graph G are summarized by its skeleton and by its v-structures (Pearl, 1988). Consequently, two DAGs having the same skeleton and v-structures entail the same set of independencies; they are said to be *equivalent* (Neapolitan, 2003). This equivalence relation defines *equivalent classes* over space of DAGs that are unambiguously represented by *completed partially directed acyclic graphs (CPDAG)* (Chickering, 2002b). Finally, if all and only the conditional independencies true in a distribution P are entailed by the Markov condition applied to a DAG G, we say that the Bayesian Network (G, P) is *faithful* (Spirtes et al., 2000).

In our case, we will assume that the probability distribution P_0 over the set of random variables $\mathbf{X}$ is faithful, that is, that there exists a graph G_0, such that (G_0, P_0) is a faithful Bayesian network. Although there are distributions P that do not admit a faithful BN (for example the case when parents are connected to a node via a parity or XOR structure), such cases are regarded as "rare" (Meek, 1995), which justifies our hypothesis.

To study $\mathbf{X}$, we are given a set of data $\mathbf{D}$ following the distribution P_0, and we try to learn a graph G, such that (G, P_0) is a faithful Bayesian network. The graph we are looking for is probably not unique because any member of its equivalent class will also be correct; however, the corresponding CPDAG is unique. Since there may be numerous graphs G to which P_0 is faithful, several definitions are possible for the problem of learning a BN. We choose as Neapolitan (2003):

Definition 2. *Let P_0 be a faithful distribution and* $\mathbf{D}$ *be a statistical sample following it. The problem of learning the structure of a Bayesian network given* $\mathbf{D}$ *is to* induce a graph G so that (G, P_0) is a faithful BN, *that is, G and G_0 are on the same equivalent class, and both are called the true structure of the system studied.*

In every IT-based or constraint-based algorithm, the following theorem is useful to identify the skeleton of G_0:

Theorem 1. (Spirtes et al., 2000) *In a faithful BN (G, P) on variables* $\mathbf{V}$*, there is an edge between the pair of nodes X and Y if and only if X depends on Y conditioning on every subset* $\mathbf{Z}$ *included in* $\mathbf{V} \setminus \{X, Y\}$.

Thus, from the data, we can estimate the skeleton of G_0 by performing conditional independency tests (Glymour and Cooper, 1999; Cheng et al., 2002). We will return to this point in Section 4.1 since higher significance levels for the test could be used to obtain a cover of the skeleton of the true graph.

3.2 General Optimal Search

Before presenting our algorithm, we should review the functioning of an OS. Among the few articles on optimal search (Ott et al., 2004; Koivisto and Sood, 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006), Ott and Miyano (2003) are to our knowledge the first to have published an exact algorithm. In this section we present the algorithm of Ott et al. (2004) for summarizing the main idea of OS. While investigating the problem of exact model averaging, Koivisto and Sood (2004) independently proposed another algorithm that also learn optimal graphs proceeding on a similar way. As for Singh and Moore (2005), they presented a recursive implementation that is less efficient in terms of calculation; however, it has the advantage that potential branch-pruning rules can be applied. Finally, Silander and Myllymäki (2006) detailed a practically efficient implementation of the search: the main advantage of their algorithm is to calculate efficiently the scores by using contingency tables (still computational complexity remains the same). They empirically demonstrated that optimal graphs could be learned up to $n = 29$.

To understand how OS finds global optima in $O(n2^n)$ without having to explicitly check every DAG possible, we must first explain how a score function is defined. Various scoring criteria for graphs have been defined, including Bayesian Dirichlet (specifically BDe with uniform priors, BDeu) (Heckerman et al., 1995), Bayesian information criterion (BIC) (Schwartz, 1978), Akaike information criterion (AIC) (Akaike, 1974), minimum description length (MDL) (Rissanen, 1978),

and Bayesian network and nonparametric regression criterion (BNRC) (Imoto et al., 2002). They are usually costly to evaluate; however, due to the Markov condition, they can be evaluated locally:

$$Score(G,\mathbf{D}) = \sum_{i=1}^{n} score(X_i, \mathbf{Pa}_i, \mathbf{D}).$$

This property is essential to enable efficient calculation, particularly with large graphs, and is usually supposed while defining an algorithm. Another classical attribute is score equivalence, which means that two equivalent graphs will have the same score. It was proved to be the case for BDe, BIC, AIC, and MDL (Chickering, 1995). In our study, we will use BIC, thereby our score is local and equivalent, and our task will be to find a DAG over $\mathbf{X}$ that maximizes the score given the data $\mathbf{D}$. Exploiting score locality, Ott et al. (2004) defined for every node X_i and every candidate parent set $\mathbf{A} \subseteq \mathbf{X} \setminus \{X_i\}$:

- The best local score on X_i: $F_s(X_i,\mathbf{A}) = \max_{\mathbf{B}\subseteq\mathbf{A}} score(X_i,\mathbf{B},\mathbf{D})$;
- The best parent set for X_i: $F_p(X_i,\mathbf{A}) = \operatorname{argmax}_{\mathbf{B}\subseteq\mathbf{A}} score(X_i,\mathbf{B},\mathbf{D})$.

From now we omit writing $\mathbf{D}$ when referring to the score function. F_s can be calculated recursively on the size of $\mathbf{A}$ using the following formulas:

$$F_s(X_i,\emptyset) = score(X_i,\emptyset), \quad (1)$$
$$F_s(X_i,\mathbf{A}) = \max(score(X_i,\mathbf{A}), \max_{X_j\in\mathbf{A}}(F_s(X_i,\mathbf{A}\backslash\{X_j\})). \quad (2)$$

Calculation of F_p directly follows; we will sometimes use F as a shorthand to refer to these two functions. Noticing that we can dynamically evaluate F, one can think that it is thus directly possible to find the best DAG. However, it is also essential to verify that the graph obtained is acyclic and hence, that there exists a topological ordering over the variables.

Definition 3. *Let w be an ordering defined on $\mathbf{A} \subseteq \mathbf{X}$ and $H = (\mathbf{A},\mathbf{E})$ be a DAG. We say that H is* w-linear *if and only if $w(X_i) < w(X_j)$ for every directed edge $(X_i,X_j) \in \mathbf{E}$.*

By using F_p and given an ordering w on $\mathbf{A}$ we derive the best w-linear graph G^*_w as:

$$G^*_w = (\mathbf{A},\mathbf{E}^*_w), \text{ with } (X_j,X_i) \in \mathbf{E}^*_w \text{ if and only if } X_j \in F_p(X_i, \mathbf{Pred}_w(X_i)). \quad (3)$$

Here, G^*_w is directly obtained by selecting for each variable $X_i \in \mathbf{A}$ its best parents among the nodes preceding X_i in the ordering w referred as $\mathbf{Pred}_w(X_i) = \{X_j \text{ with } w(X_j) < w(X_i)\}$. Therefore, to achieve OS, we need to find an optimal w^*, that is, a topological ordering of an optimal DAG. With this end, we define for every subset $\mathbf{A} \subseteq \mathbf{X}$ not empty:

- The best score of graphs G on $\mathbf{A}$: $M_s(\mathbf{A}) = \max_G Score(G)$
- The last node of an optimal ordering on $\mathbf{A}$: $M_l(\mathbf{A})$

Another way to interpret $M_l(\mathbf{A})$ is as a sink of an optimal graph on $\mathbf{A}$, that is, a node that has no children. M_s and M_l are simply initialized by:

$$\begin{aligned} \forall X_i \in \mathbf{X}: \;\; & M_s(\{X_i\}) = score(X_i, \emptyset), \\ & M_l(\{X_i\}) = X_i. \end{aligned} \tag{4}$$

When $|\mathbf{A}| = k > 1$, we consider an optimal graph G^* on that subset and w^* one of its topological ordering. The parents of the last element X_{i^*} are for sure $F_p(X_{i^*}, \mathbf{B}_{i^*})$, where $\mathbf{B}_j = \mathbf{A} \setminus \{X_j\}$; thus its local score is $F_s(X_{i^*}, \mathbf{B}_{i^*})$. Moreover, the subgraph of G^* induced when removing X_{i^*} must be optimal for $\mathbf{B}_{i^*}$; thus, its score is $M_s(\mathbf{B}_{i^*})$. Therefore, we can derive a formula to define M_l recursively:

$$M_l(\mathbf{A}) = X_{i^*} = \underset{X_j \in \mathbf{A}}{\operatorname{argmax}}(F_s(X_j, \mathbf{B}_j) + M_s(\mathbf{B}_j)). \tag{5}$$

This also enables us to calculate M_s directly. We will use M to refer to both M_s and M_l. M can be computed dynamically and M_l enables us to build quickly an optimal ordering w^*; elements are find in reverse order:

$$\begin{aligned} & \mathbf{T} = \mathbf{X} \\ & \text{While } \mathbf{T} \neq \emptyset \\ & \qquad w^*(M_l(\mathbf{T})) = |\mathbf{T}| \\ & \qquad \mathbf{T} = \mathbf{T} \setminus M_l(\mathbf{T}) \end{aligned} \tag{6}$$

Therefore, the OS algorithm is summarized by:

Algorithm 1 (OS). (Ott et al., 2004)

(a) Initialize $\forall X_i \in \mathbf{X}$, $F_s(X_i, \emptyset)$ and $F_p(X_i, \emptyset)$ with (1)

(b) For each $X_i \in \mathbf{X}$ and each $\mathbf{A} \subseteq \mathbf{X} \setminus \{X_i\}$:
Calculate $F_s(X_i, \mathbf{A})$ and $F_p(X_i, \mathbf{A})$ using (2)

(c) Initialize $\forall X_i$, $M_s(\{X_i\})$ and $M_l(\{X_i\})$ using (4)

(d) For each $\mathbf{A} \subseteq \mathbf{X}$ with $|\mathbf{A}| > 1$:
Calculate $M_s(\mathbf{A})$ and $M_l(\mathbf{A})$ using (5)

(e) Build an optimal ordering w^* using (6)

(f) Return the best w^*-linear graph $G^*_{w^*}$ using (3)

Note that in steps (b) and (d) subsets $\mathbf{A}$ are implicitly considered by increasing size to enable formulae (2) and (5). With respect to computational complexity, in steps (a) and (b) F is calculated for $n2^{n-1}$ pairs of variable and parent candidate set. In each case, one score exactly is computed. Then, M is computed over the 2^n subsets of $\mathbf{X}$ (step (c) and (d)). w^* and G^*_w are both build in $O(n)$ time at step (e) and (f); thus, the algorithm has a total time complexity of $O(n2^n)$ and evaluates $n2^{n-1}$ scores. Here, time complexity refers to the number of times that the formulae (2) or (5) are computed; however, it should be pointed out that these formulae require at least $O(n)$ basic operations.

As proposed (Ott et al., 2004), OS can be speed up by constraining with a constant c the maximal size of parent sets. This limitation is easily justifiable, as graphs having many parents for a node are usually strongly penalized by score functions. In that case, the computational complexity remains the same; only formulas (2) is constrained, and $score(X_i, \mathbf{A})$ is not calculated when $|\mathbf{A}| > c$. Consequently, the total number of score evaluated is reduced to $O(n^{c+1})$, which is a decisive improvement since computing a score is costly.

The space complexity of Algorithm 1 can be slightly reduced by recycling memory as mentioned (Ott et al., 2004). In fact, when calculating functions F and M for subsets $\mathbf{A}$ of size k, only values for subsets of size $k-1$ are required. Therefore, by computing simultaneously these two functions, when values for subsets of a given size have been computed, the memory used for smaller set can be reused. However, to be able to access G^*_w, we should redefine M_l to store optimal graphs instead of optimal sinks. The worst memory usage corresponds to $k = \lfloor \frac{n}{2} \rfloor + 1$ when we have to consider approximately $O(\frac{2^n}{\sqrt{n}})$ sets: this approximation comes from Stirling formula applied to the binomial coefficient of n and $\lfloor \frac{n}{2} \rfloor$ ($\lfloor x \rfloor$ is the highest integer less than or equal to x). At that time, $O(\sqrt{n}2^n)$ best parent sets are stored by F, and $O(\frac{2^n}{\sqrt{n}})$ graphs by M. Since a parent set requires $O(n)$ space and a graph $O(n^2)$, we derive that the maximal memory usage with recycling is $O(n^{\frac{3}{2}}2^n)$, while total memory usage of F in Algorithm 1 was $O(n^2 2^n)$. Actually, since Algorithm 1 is feasible only for small n, we can consider that a set requires $O(1)$ space (represented by less than k integers on a x-bit CPU if $n < kx$): in that case also, the memory storage is divided by a factor $\sqrt{n}$ with recycling.

Ott et al. (2005) also adapted their algorithm to list as many suboptimal graphs as desired. Such capacity is precious in order to find which structural similarities are shared by highly probable graphs, particularly when the score criteria used is not equivalent. However, for an equivalent score, since the listed graphs will be mainly on the same equivalent classes, they will probably not bring more information than the CPDAG of an optimal graph.

4. Super-Structure Constrained Optimal Search

Compare to a brute force algorithm that would browse all search space, OS achieved a considerable improvement. Graphs of around thirty nodes are still hardly computed, and many small real networks such as the classical ALARM network (Beinlich et al., 1989) with 37 variables are not feasible at all. The question of an optimal algorithm with a lower complexity is still open. In our case, we focus on structural constraint to reduce the search space and develop a faster algorithm.

4.1 Super-Structure

To keep the property that the result of OS converges to the true graph in the sample limit, the constraint should at least authorize the true skeleton. Since knowing the true skeleton is a strong assumption and learning it with high confidence from finite data is a hard task, we propose to consider a more flexible constraint than fixing the skeleton. To this end, we introduce a super-structure as:

Definition 4. *An undirected graph $S = (\mathbf{V}, \mathbf{E}_S)$ is said to be a* super-structure *of a DAG $G = (\mathbf{V}, \mathbf{E}_G)$, if the skeleton of G, $G' = (\mathbf{V}, \mathbf{E}_{G'})$ is a subgraph of S (i.e., $\mathbf{E}_{G'} \subseteq \mathbf{E}_S$). We say that S* contains *the skeleton of G.*

Considering a structure learning task, a super-structure S is said to be true or *sound* if it contains the true skeleton; otherwise it is said *incomplete*. Finally we propose to study the problem of model

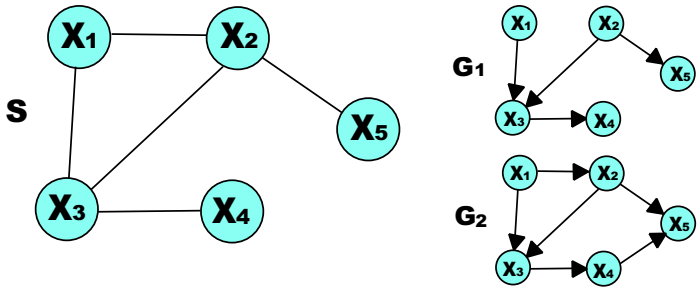


Figure 1: In a search constrained by S, G_1 could be considered but not G_2 because $\langle X_4, X_5 \rangle \notin \mathbf{E}_S$.

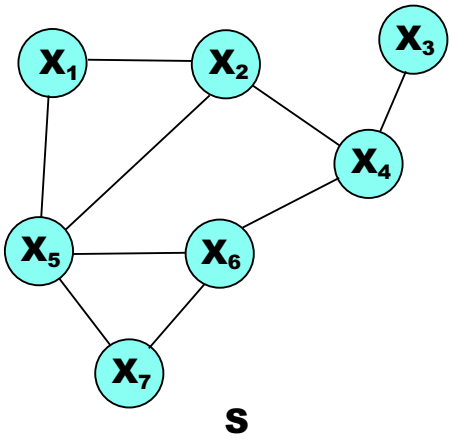
inference from data given a super-structure S: S is assumed to be sound, and the search space is restricted to DAGs whose skeletons are contained in S as illustrated in Figure 1. Actually, the "skeleton" learned by MMPC is used as a super-structure in MMHC. In fact, the skeleton of the graph returned by MMHC is not proven to be the same than the learned one; some edges can be missing. It is the same for the candidate parents in SC. Thus, the idea of super-structure already existed, but we define it explicitly, which has several advantages.

First, a dedicated terminology enables to emphasize two successive and independent phases in structure learning problem: on one hand, learning with high probability a sound super-structure S (sparse if possible); on the other hand, given such structure, searching efficiently the restricted space and returning the best optimum found (global optimum if possible). This problem cutting enables to make clearer the role and effect of each part. For example, since SC and MMHC use the same search, comparing their results allow us directly to evaluate their super-structure learning approach. Moreover, while conceiving a search strategy, it could be of a great use to consider a super-structure given. This way, instead of starting from a general intractable case, we have some framework to assist reasoning: we give some possible directions in our future work. Finally, this manner to apprehend the problem already integrates the idea that the true skeleton will not be given by an IT approach; hence, it could be better to learn a bit denser super-structure to reduce missing edges, which should improve accuracy.

Finally, we should explain how practically a sound super-structure S can be inferred. Even without knowledge about causality, a quick analysis of the system could generate a rough draft by determining which interactions are impossible; localization, nature or temporality of variables often forbid evidently many edges. In addition, for any IT-based technique to learn the skeleton, the neighborhood of variables or their Markov blanket could be used to get a super-structure. This one should become sound while increasing the significance level of the tests: this is because we only need to reduce false negative discovery. Although the method used in PC algorithm could be a good candidate to learn a not sparse but sound super-structure, we illustrate our idea with MMPC in Section 5.1.

4.2 Constraining and Optimizing

From now on, we will assume that we are given a super-structure $S = (\mathbf{X}, \mathbf{E}_S)$ over $\mathbf{X}$. We refer to the neighborhood of a variable X_i in S by $\mathbf{N}(X_i)$, that is, the set of nodes connected to X_i in S (i.e., $\{X_j \mid \langle X_i, X_j \rangle \in \mathbf{E}_S\}$); m is the maximal degree of S, that is, $m = \max_{X_i \in \mathbf{X}} |\mathbf{N}(X_i)|$. Our task is to globally maximize the score function over our reduced search space.



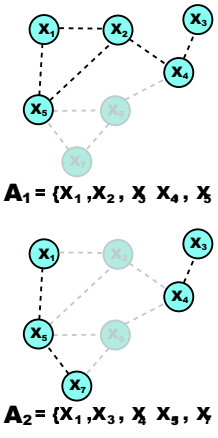


Figure 2: $\mathbf{A}_1$ is in $Con(S)$, but not $\mathbf{A}_2$ because in $S_{\mathbf{A}_2}$, X_7 and X_4 are not connected.

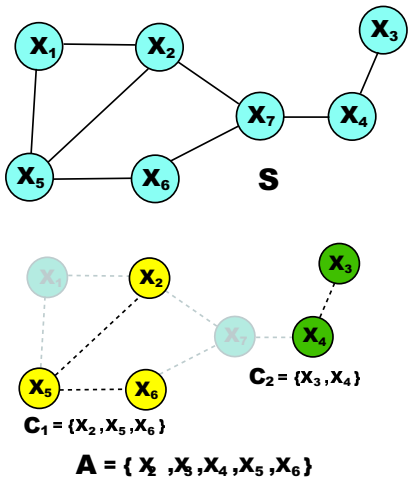


Figure 3: The maximal connected subsets of $\mathbf{A}$: $\mathbf{C}_1$ and $\mathbf{C}_2$.

Since the parents of every X_i are constrained to be included in $\mathbf{N}(X_i)$, the function F has to be defined only for $\forall \mathbf{A} \subseteq \mathbf{N}(X_i)$. Consequently, computation of F in step (b) becomes:

(b*) For each $X_i \in \mathbf{X}$ and each $\mathbf{A} \subseteq \mathbf{N}(X_i)$
Calculate $F_s(X_i, \mathbf{A})$ and $F_p(X_i, \mathbf{A})$ using (2)

Only the underlined part has been modified; clearly, F still can be computed recursively since $\forall X_j \in \mathbf{A}$, the subset $\mathbf{A} \backslash \{X_j\}$ is also included in $\mathbf{N}(X_i)$, and its F value is already known. With this slight modification, the time complexity of computing F becomes $O(n2^m)$, which is a decisive improvement opening many perspectives; more details are given at the end of this section. However, to keep formulae (5) and (3) correct, $F(X_i, \mathbf{A})$ for any subset $\mathbf{A}$ has to be replaced by $F(X_i, \mathbf{A} \cap \mathbf{N}(X_i))$. Before simplifying the calculation of M, it is necessary to introduce the notion of *connectivity*:

Definition 5. *Given an undirected graph $S = (\mathbf{X}, \mathbf{E}_S)$, a subset $\mathbf{A} \subseteq \mathbf{X}$ is said to be* connected *if $\mathbf{A} \neq \emptyset$ and the subgraph of S induced by $\mathbf{A}$, $S_{\mathbf{A}}$, is connected (cf. Figure 2).*

In our study, connectivity will always refer to the connectivity in the super-structure S. $Con(S)$ will refer to the set of connected subsets of $\mathbf{X}$. In addition, each not empty subset of $\mathbf{X}$ can be broken down uniquely into the following family of connected subsets:

Definition 6. *Given an undirected graph $S = (\mathbf{X}, \mathbf{E}_S)$ and a subset $\mathbf{A} \subseteq \mathbf{X}$, let $S_1 = (\mathbf{C}_1, \mathbf{E}_1), \cdots, S_p = (\mathbf{C}_p, \mathbf{E}_p)$ be the connected components of the induced subgraph $S_{\mathbf{A}}$. The subsets $\mathbf{C}_1, \cdots, \mathbf{C}_p$ are called the* maximal connected subsets *of $\mathbf{A}$ (cf. Figure 3).*

The most important property of the maximal connected subsets $\mathbf{C}_1, \cdots, \mathbf{C}_p$ of a subset $\mathbf{A}$ is that, when $p > 1$ (i.e., when $\mathbf{A} \notin Con(S)$) for any pair $\mathbf{C}_i$, $\mathbf{C}_j$ with $i \neq j$, $\mathbf{C}_i \cap \mathbf{C}_j = \emptyset$ and there is no edges in S between nodes of $\mathbf{C}_i$ and nodes of $\mathbf{C}_j$. Next we show that the value of M for subsets that are unconnected do not have to be explicitly calculated, which is the second and last modification

of Algorithm 1. The validity of our algorithm is simultaneously proved.

Theorem 2. *A constrained optimal graph can be found by computing M only over Con*(S).
Proof: First, let consider a subset $\mathbf{A} \notin Con(S)$, its maximal connected subsets $\mathbf{C}_1, \cdots, \mathbf{C}_p$ ($p > 1$), and an optimal constrained DAG $G^* = (\mathbf{A}, \mathbf{E}^*)$. Since G^* is constrained by the super-structure, and following the definition of the maximal connected subsets, there cannot be edges in G^* between any element in $\mathbf{C}_i$ and any element in $\mathbf{C}_j$ if $i \neq j$. Therefore, the edges of G^* can be divided in p sets $\mathbf{E} = \mathbf{E}_1 \cup \cdots \cup \mathbf{E}_p$ with $G_i = (\mathbf{C}_i, \mathbf{E}_i)$ a DAG over every $\mathbf{C}_i$. Moreover, all G_i are optimal constrained graphs otherwise G^* would not be. Consequently, we can derive the two following formulas:

$$M_s(\mathbf{A}) = \sum_{i=1}^{p} M_s(\mathbf{C}_i), \tag{7}$$

$$M_l(\mathbf{A}) = M_l(\mathbf{C}_1). \tag{8}$$

Formula (7) directly follows our previous considerations that maximizing the score over $\mathbf{A}$ is equivalent to maximizing it over each $\mathbf{C}_i$ independently, since they cannot affect each other. Actually, any $M_l(\mathbf{C}_i)$ is an optimal sink and could be selected in (8); we chose $M_l(\mathbf{C}_1)$ since it is accessed faster when using the data structure proposed in Section 4.3 for M. By using (7) and (8) the value of M for unconnected subsets can be directly computed if needed from the values of smaller connected subsets. Therefore, we propose to compute M only for connected subsets by replacing step (d) with (d*) in Algorithm 2 described below. Since each singleton $\{X_i\}$ is connected, step (c) is not raising a problem. In step (d*) we consider $\mathbf{A} \in Con(S)$ and apply formula (5), if there is $X_j \in \mathbf{A}$ such that $\mathbf{B}_j = \mathbf{A} \backslash \{X_j\}$ is not connected, we then can directly calculate $M_s(\mathbf{B}_j)$ by applying (7). the values of M_s for the maximal connected subsets of $\mathbf{B}_j$ are already computed since these subsets are of smaller sizes than $\mathbf{A}$. Therefore, $M_l(\mathbf{A})$ and $M_s(\mathbf{A})$ can be computed. Finally, it is also possible to retrieve w^* from (6) by using (8) if $\mathbf{T}$ is not connected, which conclude the proof of this Theorem. ■

We can now formulate our optimized version of Algorithm 1 for optimal DAG search conditioned by a super-structure S:

Algorithm 2.

(a*) Initialize $\forall X_i \in \mathbf{X}$, $F_s(X_i, \emptyset)$ and $F_p(X_i, \emptyset)$ with (1)

(b*) For each $X_i \in \mathbf{X}$ and each $\mathbf{A} \subseteq \mathbf{N}(X_i)$
Calculate $F_s(X_i, \mathbf{A})$ and $F_p(X_i, \mathbf{A})$ using (2)

(c*) Initialize $\forall X_i$, $M_s(\{X_i\})$ and $M_l(\{X_i\})$ using (4)

(d*) For each $\mathbf{A} \in Con(S)$ with $|\mathbf{A}| > 1$
Calculate $M_s(\mathbf{A})$ and $M_l(\mathbf{A})$ using (5) and (7)

(e*) Build an optimal ordering w^* using (6) and (8)

(f*) Return the best w^*-linear graph $G^*_{w^*}$ using (3)

The underlined parts of Algorithm 2 are the modifications introduced in Algorithm 1. Computational complexity and correctness of (b*) has already been presented. With Theorem 2, validity of our algorithm is assured and since in (c*) and (d*) every element of $Con(S)$ are considered only

once, the total computational complexity is in $O(n2^m + |Con(S)|)$; here again complexity refers to the number of times formulae (2) or (5) are computed. We will describe in the next Section a method to consider only connected subsets, and come over the number of connected subsets of S in Section 4.4. Although set operators are used heavily in our algorithm, such operations can be efficiently implemented and considered of a negligible cost as compared to other operations, such as score calculations. Concerning the complexity of calculating F, $O(n2^m)$ is in fact a large upper bound. Still, since it depends only linearly on the size of the graphs, F can be computed for graphs of any size if their maximal degree is less than around thirty. This enables usage of this function in new search strategies for many real systems that could not be considered without constraint. We should remark that some cases of interest still cannot be studied since this upper limitation on m constrains also the maximal number of children of every variables. However, this difficulty concerns also many IT-approaches since their complexity also depends exponentially on m (see Tsamardinos et al. 2006 for MMPC and Kalisch and Bühlmann 2007 for PC). Finally, like in Algorithm 1, the number of scores calculated can be reduced to $O(nm^c)$ by constraining on the number of parents.

Although the number of M and F values calculated is strictly reduced, a potential drawback of Algorithm 2 is that memory cannot be recycled anymore. First, when (5) is used during step (d*), now $F_s(X_i, \mathbf{B}_j \cap \mathbf{N}(X_i))$ is required, and nothing can be said about $|\mathbf{B}_j \cap \mathbf{N}(X_i)|$ implying that we should store every value of F_s computed before. Similar arguments hold for F_p in case M_l is used to store optimal graphs, and for M_s and M_l because (7) and (8) could have to be used anytime during (d*) and (e*) respectively. However, since space complexity of F is $O(n2^m)$ and the one of M is $O(|Con(S)|)$ (cf. next section), if m is bounded Algorithm 2 should use less memory than Algorithm 1 even when recycling memory (i.e., $O(\sqrt{n}2^n)$ assuming that a set takes $O(1)$ space). This soften the significance of recycling memory in our case.

Finally, since our presentation of Algorithm 2 is mainly formal, we should detail how it is practically possible to browse efficiently only the connected subsets of $\mathbf{X}$. For this, we present in the next section a simple data structure to store values of M and a method to build it.

4.3 Representation of $Con(S)$

For every $\mathbf{A} \in Con(S)$ we define $\mathbf{N}(\mathbf{A}) = \left(\bigcup_{X_i \in \mathbf{A}} \mathbf{N}(X_i)\right) \setminus \mathbf{A}$, that is the set of variables neighboring $\mathbf{A}$ in S. For every $X_i \notin \mathbf{A}$, we note $\mathbf{A}_i^+ = \mathbf{A} \cup \{X_i\}$, it is connected if and only if $X_i \in \mathbf{N}(\mathbf{A})$. Finally, for a subset $\mathbf{A}$ not empty, let $min(\mathbf{A})$ be the smallest index in $\mathbf{A}$, that is, $min(\mathbf{A}) = i$ means that $X_i \in \mathbf{A}$ and $\forall X_j \in \mathbf{A}$, $j \geq i$; by convention, $min(\emptyset) = 0$. Now we introduce an auxiliary directed graph $G^\star = (Con(S)^\star, \mathbf{E}^\star)$, where $Con(S)^\star = Con(S) \cup \{\emptyset\}$ and the set of directed edges $\mathbf{E}^\star$ is such that there is an edge from $\mathbf{A}$ to $\mathbf{B}$ if and only if $\mathbf{A} \subset \mathbf{B}$ and $|\mathbf{B}| = |\mathbf{A}| + 1$. In other words, with the convention that $\mathbf{N}(\emptyset) = \mathbf{X}$, $\mathbf{E}^\star = \{(\mathbf{A}, \mathbf{A}_i^+), \forall \mathbf{A} \in Con(S)^\star \text{ and } \forall X_i \in \mathbf{N}(\mathbf{A})\}$. Actually, $G^\star$ is trivially a DAG since arcs always go from smaller to bigger subsets. Finally, let define H as being the spanning tree obtained from the following depth-first-search (DFS) on $G^\star$:

- The search starts from $\emptyset$.
- While visiting $\mathbf{A} \in Con(S)^\star$: for all $X_{k_i} \in \mathbf{N}(\mathbf{A})$ considered by increasing indices (i.e., such that $k_1 < \cdots < k_p$, where $p = |\mathbf{N}(\mathbf{A})|$) visit the child $\mathbf{A}_{k_i}^+$ if it was not yet done. When all children are done, the search backtracks.

Since there is a path from the empty set to every connected subset in $G^\star$, the nodes of the tree H represent unambiguously $Con(S)^\star$. We use H as a data structure to store the values of M in

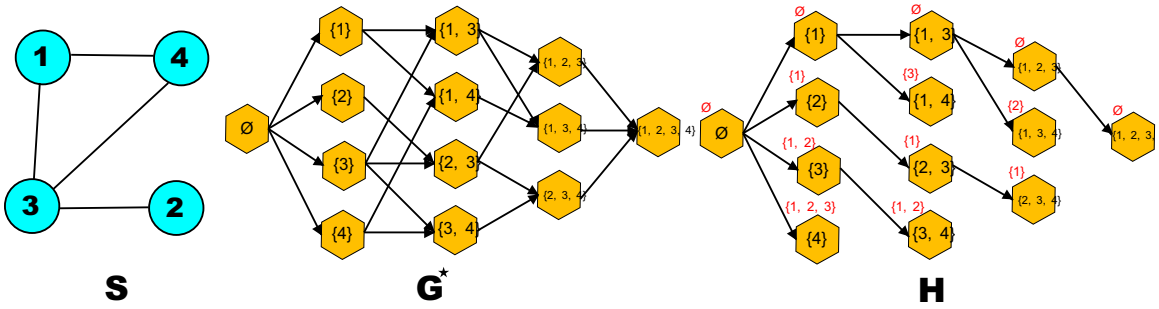


Figure 4: $G^\star$ and H for a given S. $\mathbf{Fb}(\mathbf{A})$ is indicated in red above each node in H.

Algorithm 2; this structure is illustrated by an example in Figure 4. Further, we propose a method to build H directly from S without having to build $G^\star$ explicitly. First, we notice that after visiting $\mathbf{A}$, every $\mathbf{B} \in Con(S)$ such that $\mathbf{B} \supset \mathbf{A}$ have been visited for sure. When visiting $\mathbf{A}$, we should consider only the children of $\mathbf{A}$ that were not yet visited. For this, we define:

- $\mathbf{Fb}(\mathbf{A})$ is the set of forbidden variables of $\mathbf{A}$, that is: for every $\mathbf{B} \in Con(S)$ with $\mathbf{B} \supset \mathbf{A}$, $\mathbf{B}$ has been already visited if and only if $\mathbf{B} \supseteq \mathbf{A}^+_j$ with $X_j \in \mathbf{Fb}(\mathbf{A})$.

By defining recursively this forbidden set for every child of $\mathbf{A}$ that has not yet been visited, we derive the following method to build H:

Method 1.
1: Create the root of H (i.e., $\emptyset$), and initialize $\mathbf{Fb}(\emptyset) = \emptyset$.
2: For i from 0 to $n-1$, and for all $\mathbf{A}$ in H such that $|\mathbf{A}| = i$
3: Set $\mathbf{Fb}^* = \mathbf{Fb}(\mathbf{A})$,
4: For every $X_{k_j} \in \mathbf{N}(\mathbf{A}) \backslash \mathbf{Fb}(\mathbf{A})$ considered by increasing indices
5: Add to $\mathbf{A}$ the child $\mathbf{A}^+_{k_j}$ in H and define $\mathbf{Fb}(\mathbf{A}^+_{k_j}) = \mathbf{Fb}^*$
6: Update $\mathbf{Fb}^* = \mathbf{Fb}^{*+}_{k_j}$

The correctness of Method 1 is proven in the next Theorem. In order to derive the time complexity of step (d*) in terms of basic operations, while using Method 1 in Algorithm 2, let consider that the calculation takes place on a x-bit machine and that n is at maximum few times greater than x. Thus, subsets requires $O(1)$ space, and any operations on subsets are done in $O(1)$ time, except $min(\mathbf{A})$ (in $O(\log(n))$ time).

Theorem 3. *The function M can be computed in time and space proportional to $O(|Con(S)|)$, up to some polynomial factors. With Method 1, M is computed in $O(\log(n)n^2|Con(S)|)$ time and requires $O(|Con(S)|)$ space.*

Proof: First, to prove correctness of Method 1, we show that if $\mathbf{Fb}(\mathbf{A})$ is correctly defined in regard to our DFS, the search from $\mathbf{A}$ proceeds as expected and that before back tracking every connected superset of $\mathbf{A}$ has been visited. The case when all the variables neighboring $\mathbf{A}$ are forbidden being trivial, we directly consider all the elements $X_{k_1}, \cdots, X_{k_p}$ of $\mathbf{N}(\mathbf{A}) \backslash \mathbf{Fb}(\mathbf{A})$ by increasing indices like in DFS (with $p \geq 1$). Then, $\mathbf{A}^+_{k_1}$ should be visited first, and $\mathbf{Fb}(\mathbf{A}^+_{k_1}) = \mathbf{Fb}(\mathbf{A})$ because $\forall X_j \in$

$\mathbf{Fb}(\mathbf{A})$ among the supersets of $\mathbf{A}_j^+$ there is also the supersets of $\mathbf{A}_{k_1,j}^{++}$. Now let suppose that the forbidden sets were correctly defined and that the visits correctly proceeded until $\mathbf{A}_{k_i}^+$: if $i = p$ by hypothesis we explored every connected supersets of the children of $\mathbf{A}$ and the search back track correctly. Otherwise, we should define $\mathbf{Fb}(\mathbf{A}_{k_{i+1}}^+) = \mathbf{Fb}(\mathbf{A}) \cup \{X_{k_1}\} \cup \cdots \cup \{X_{k_i}\} = \mathbf{Fb}(\mathbf{A}_{k_i}^+)_{k_i}^+$ to take into account all supersets visited during the previous recursive searches. Although Method 1 does not proceed recursively (to follow the definition of M), since it uses the same formulae to define the forbidden sets, and since $\mathbf{Fb}(\emptyset)$ is correctly initialized, H is built as expected.

To be able to access easily $M(\mathbf{A})$, we keep for every node an auxiliary set defined by $\mathbf{Nb}(\mathbf{A}) = \mathbf{N}(\mathbf{A}) \setminus \mathbf{Fb}(\mathbf{A})$ that is easily computed as processing Method 1. Since there is $O(|Con(S)|)$ nodes in H, each storing a value of M and two subsets requiring $O(1)$ space, the assertion about space complexity is correct.

Finally, building a node requires $O(1)$ set operations. To access $M(\mathbf{A})$ even for an unconnected subset, we proceed on the following manner: we start from the root, and define $\mathbf{T} = \mathbf{A}$. Then, when we rushed the node $\mathbf{B} \subseteq \mathbf{A}$, with $i = min(\mathbf{Nb}(\mathbf{B}) \cap \mathbf{T})$, we withdraw X_i from $\mathbf{T}$, and go down to the i^{th} child of $\mathbf{B}$. If $i = 0$ then if $\mathbf{T} = \emptyset$ we found the node of $\mathbf{A}$; otherwise we rushed the first maximal connected component of $\mathbf{A}$, that is, $\mathbf{C}_1$ of which we can accumulate the M value in order to apply (7) or (8). In that case, we continue the search by restarting from the root with the variables remaining in $\mathbf{T}$. In any cases, at maximum *min* is used $O(n)$ times to find $M(\mathbf{A})$, implying a time complexity of $O(\log(n)n^2)$ for formula (5). ■

It is interesting to notice that, even without memoization, the values of M can be calculated in different order. For example, by calling H_i the subtree of H starting from $\{X_i\}$, only values of M over H_j such that $j \geq i$ are required. Then it is feasible to calculate M from H_n to H_1, which could be used to apply some branch-pruning rules based on known values of M or to apply different strategies depending on the topology of the connected subset; these are only suppositions.

More practically, other approach could be proposed to build a spanning tree of $G^\star$ and Method 1 is presented here only to complete our implementation and illustrate Theorem 3. We should note that one can also implement the calculation of M over $Con(S)$ in a top-down fashion using memoization and an auxiliary function to list maximal connected components of a subset. However, such implementations will be less efficient both in space and time complexity. Even without considering the cost of recursive programming, listing connected components is in $O(nm)$. Then, in order to not waste an exponential factor of memory, self balanced trees should be used to store the memorized values of M: it would require $O(n)$ time to access a value and up to $O(n^2)$ if (7) is used. This should be repeated $O(n)$ times to apply (5), which implies a complexity of around $O(n^3|Con(S)|)$. Consequently, we believe that Method 1 is a competitive implementation of step (d*).

4.4 Counting Connected Subsets of a Graph

To understand the complexity of Algorithm 2, the asymptotic behavior of $|Con(S)|$ should be derived, depending on some attributes of S. Comparing the trivial cases of a linear graph (i.e., $m \leq 2$) where $|Con(S)| = O(n^2)$ and a star graph (i.e., one node is the neighbor of all others) where $|Con(S)| = 2^{n-1} + n - 1$ clearly indicates that $|Con(S)|$ depends strongly on the degrees of Srather than on the number of edges or number of cycles. One important result from Björklund et al. (2008) is that $|Con(S)| = O(\beta_m^n)$ with $\beta_m = (2^{m+1} - 1)^{\frac{1}{m+1}}$ a coefficient that only depends on the maximal degree m of S.

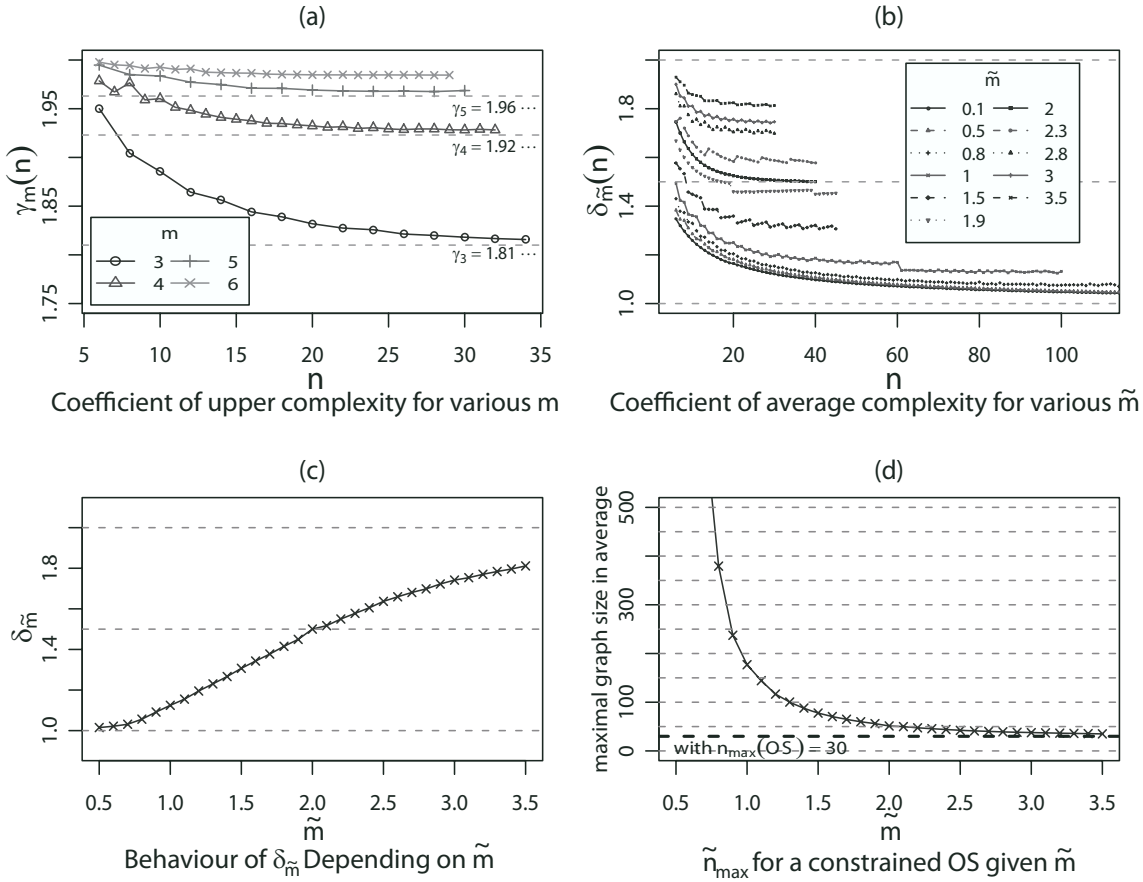


Figure 5: Experimental derivation of γ_m, $\delta_{\tilde{m}}$ and $\tilde{n}_{\max_2}(\tilde{m})$.

Still, since this upper bound is probably over-estimated, we tried to evaluate a better one experimentally. For every pair (m,n) of parameters considered , we randomly generated 500 undirected graphs S pushing their structures towards a maximization of the number of connected subsets. For this, all the nodes had exactly m neighbors and S should at least be connected. Then, since after a first series of experiments the most complex structures appeared to be similar to full $(m-1)$-trees, with a root having m children and leaves being connected to each other, only such structures were considered during a second series of experiments. Finally, for each pair (m,n), from the maximal number $R_{n,m}$ of connected subsets found, we calculated $\exp(\frac{\ln(R_{n,m})}{n})$ in order to search for an exponential behavior as shown in Figure 5(a).

Results 1. *Our experimental measures led us to propose that* $|Con(S)| = O({\gamma_m}^n)$ *(cf. Figure 5(a)).*

The weak point of our strategy is that more graphs should be consider while increasing n, since the number of possible graphs also increases. Unfortunately, this is hardly feasible in practice since counting gets longer with larger graphs. Nevertheless, Results 1 were confirmed during a more detailed study of the case $m = 3$ using 10 times more graphs and up to $n = 30$. In addition, even this estimated upper bound is practically of a limited interest since it still dramatically overestimates $|Con(S)|$ for real networks. Real networks are not, in general, as regular and as dense graphs as

Complexity	Algorithm 1 (OS)	Algorithm 2 (COS)
Time (steps)	$O(n2^n)$	$O(\|Con(S)\|)$
in details	$O(n^2 2^n)$	$O(\log(n)n^2\|Con(S)\|)$
scores computed	$n2^n$	$O(n2^m)$
if $\|\mathbf{Pa}_i\| < c$	$O(n^{c+1})$	$O(nm^c)$
Space	$O(\sqrt{(}n)2^n)$	$O(\|Con(S)\|)$

Table 1: Improvement achieved by COS.

S	$\|Con(S)\|$	some values
Tree-like	$O(\alpha_m^n)$	$\alpha_3 \approx 1.58$, $\alpha_4 \approx 1.65$, $\alpha_5 \approx 1.707$
General	$O(\beta_m^n)$	$\beta_3 \approx 1.968$, $\beta_4 \approx 1.987$, $\beta_5 \approx 1.995$
Measured	$O(\gamma_m^n)$	$\gamma_3 \approx 1.81$, $\gamma_4 \approx 1.92$, $\gamma_5 \approx 1.96$
In average	$O(\delta_{\tilde{m}}^n)$	$\delta_{1.5} \approx 1.3$, $\delta_2 \approx 1.5$, $\delta_{2.5} \approx 1.63$, $\delta_3 \approx 1.74$

Table 2: Results on $|Con(S)|$

the ones used in previous experiments. To illustrate that $O({\gamma_m}^n)$ is a pessimistic upper bound, we derived the theoretical upper bound of $|Con(S)|$ for tree-like structures of maximal degree m. This is given as an example, although it might help estimation of $|Con(S)|$ for structures having a bounded number of cycles. The proof is deferred to the Appendix.

Proposition 1. *If S is a forest, then $|Con(S)| = O(\alpha_m^n)$ with $\alpha_m = (\frac{2^{m-1}+1}{2})^{\frac{1}{m-1}}$.*

Finally, we studied the average size of $|Con(S)|$ for a large range of average degrees $\tilde{m}$. For each pair $(n,\tilde{m})$ considered, we generated 10000 random graphs and averaged the number of connected subsets to obtain $R_{n,\tilde{m}}$. No constraint was imposed on m, since graphs were generated by randomly adding edges until $\lfloor \frac{n\tilde{m}}{2} \rfloor$. In each case, we calculated $\exp(\frac{\ln(R_{n,\tilde{m}})}{n}))$ to search for an asymptotic behavior on $\tilde{m}$.

Results 2. *On average, for super-structures having an average degree of $\tilde{m}$, $|Con(S)|$ increases asymptotically as $O({\delta_{\tilde{m}}}^n)$, see Figure 5(b) and (c) for more details.*

Based on the assumption that Algorithm 1 is feasible at maximum for graphs of size $n_{\max_1} = 30$ (Silander and Myllymäki, 2006), we calculated $\tilde{n}_{\max_2}(\tilde{m}) = n_{\max_1}\frac{\ln(2)}{\ln(\delta_{\tilde{m}})}$ that can be interpreted as an estimation of the maximal size of graphs that can be considered by Algorithm 2 depending on $\tilde{m}$. As shown in Figure 5(d), on average, it should be feasible to consider graphs having up to 50 nodes with Algorithm 2 if $\tilde{m} = 2$. Moreover, since $\lim_{\tilde{m}\to 0} \delta_{\tilde{m}} = 1$, our algorithm can be applied to graphs of any sizes if they are enough sparse. Unfortunately, the case when $\tilde{m} < 2$ is not really interesting since it implies that networks are mainly unconnected.

To conclude, we summarize and compare the time and space complexities of both Algorithms in Table 1, using the same hypothesis on the size of a subset as Section 4.3. We neglected the complexity due to F in our algorithm, which is justified if m is not huge. Concerning the space complexity of Algorithm 1, the maximal space needed while recycling memory is used. Results on $O(|Con(S)|)$ are listed in Table 2. If super-structures S are relatively sparse and have a bounded maximal degree, the speed improvement of Algorithm 2 over Algorithm 1 should increase expo-

nentially with n. Moreover, our algorithm can be applied to some small real networks that are not feasible for Algorithm 1.

5. Experimental Evaluation

Although the demonstrations concerning correctness and complexity of Algorithm 2 enable us to anticipate results obtained experimentally, some essential points remain to be studied. Among them, we should demonstrate practically that, in the absence of prior knowledge, it is feasible to learn a sound super-structure with a relaxed IT approach. We choose to test our proposition on MMPC (Tsamardinos et al., 2006) in Section 5.2; the details of this algorithm are briefly reintroduced in Section 5.1.4. Secondly, we compare COS to OS to confirm the speed improvement of our method and study the effect of using a sound constraint in Section 5.3. We also should evaluate the worsening in terms of accuracy due to the incompleteness of an approximated super-structure. In this case we propose and evaluate a greedy method, COS+, to improve substantially the results of COS. In Section 5.4, we compare our methods to other greedy algorithms to show that, even with an incomplete super-structure, COS, and especially COS+, are competitive algorithms to study small networks. Finally we illustrate this point by studying the ALARM Network (Beinlich et al., 1989) in Section 5.5, a benchmark of structure learning algorithm for which OS is not feasible.

5.1 Experimental Approach

Except in the last real experiment, we are interested in comparing methods or algorithms for various set of parameters, such as: the size of the networks n, their average degree $\tilde{m}$ (to measure effect of sparseness on Algorithm 2), the size of data considered d and the significance level α used to learn the super-structure.

5.1.1 NETWORKS AND DATA CONSIDERED

Due to the size limitation imposed by Algorithm 1, only small networks can be learned. Further, since there it is hardly feasible to find many real networks for every pair $(n,\tilde{m})$ of interest, we randomly generated the networks to which we apply structure learning. Given a pair $(n,\tilde{m})$, DAGs of size n are generated by adding randomly $\lfloor \frac{n\tilde{m}}{2} \rfloor$ edges while assuring that cycles are not created.

For simplicity, we considered only Boolean variables; therefore, Bayesian networks are obtained from each DAG by generating conditional probabilities $P(X_i = 0|\mathbf{Pa}_i = \mathbf{pa}_i^k)$ for all X_i and all possible $\mathbf{pa}_i^k$ by choosing a random real number in $]0,1[$. Then, d data are artificially generated from such Bayesian networks, by following their entailed probability distribution.

Finally, the data are used to learn a network with every algorithm and some criteria are measured. In order to generalize our results, we repeat g times the previous steps for each quadruplet $(n,\tilde{m},\alpha,d)$. The values of each criterion of comparison for every algorithm are averaged on the g learned graphs.

5.1.2 COMPARISON CRITERIA

While learning Bayesian networks, we evaluate the performances of every algorithm on three criteria. Since the learning task consists in the maximization of the score function, a natural criterion to evaluate the quality of the learned network is its score. In our experiments we use BIC because of its speed of evaluation. Since we are interested in comparing results in terms of score depending on

n or $\tilde{m}$ in a diagram, we do not directly represent scores (their values change radically for different parameters) but use a score ratio to the optimal score: $\frac{Score(G_{OS})}{Score(G_{\text{Other}})}$, where the label of the graph indicates which Algorithm was used. The better is the score obtained by an algorithm, the closer to 1 is its score ratio. We preferred to use the best score rather than the score of the true network, because the true network is rarely optimal; its score is even strongly penalized if its structure is dense and data sets are small. Therefore, it is not convenient to use it as a reference in terms of score.

The second criterion is a measure of the complexity estimated by the execution time of each algorithm, referred as *Time*. Of course, this is not the most reliable way to estimate complexity, but since calculations are done on the same machine, and since measures are averaged on few similar calculations, execution time should approximate correctly the complexity. To avoid bias of this criterion, common routines are shared among algorithms (such as the score function, the structure learning method and the hill climbing search).

Finally, since our aim is to learn a true network, we use a structural hamming distance that compares the learned graph with the original one. As proposed in Tsamardinos et al. (2006), to take into consideration equivalence classes, the CPDAGs of both original and learned DAGs are built and compared. This defines the structural error ratio SER of a learned graph, which is the number of extra, missing, and wrongly oriented edges divided by the total number of edges in the original graph. In our case, we penalize wrongly oriented edges only by half, because we consider that errors in the skeleton are more "grave" than those in edges orientation. The reason is not only visual: a missing edge, or an extra edge, implies more mistakes in terms of conditional independencies in general than wrongly oriented ones. Moreover, in CPDAGs, the fact that an edge is not correctly oriented is often caused by extra or missing edges. Furthermore, such a modification does not intrinsically change the results, since it benefits every algorithm on the same manner.

5.1.3 HILL CLIMBING

Although hill climbing searches are used by different algorithms, we implemented only one search that is used in all cases. This search can consider a structural constraint S, and is given a graph G_{init} from which to start the search. Then it processes as summarized in Section 2.2, selecting at each step the best transformation among all edge withdrawals, edge reversals and edge additions according to the structure constraint. The search stops as soons as the score cannot be strictly improved anymore. If several transformations involve the same increase of the score, the first transformation encountered is applyed. This implies that the results will depend on the ordering of the variables; however, since the graphs considered are randomly generated, their topological ordering is also random, and in average the results found by our search should not be biased.

5.1.4 RECALL ON MMPC

In Section 5.3 a true super-structure is given as a prior knowledge; otherwise we should use an IT-approach to approximate the structural constraint S from data. Since MMHC algorithm is included in our experiments, we decided to illustrate our idea of relaxed independency testing on MMPC strategy (Tsamardinos et al., 2006).

In MMPC, the following independency test is used to infer independencies among variables. Given two variables X_i and X_j, it is possible to measure if they are independent conditioning on a subsets of variables $\mathbf{A} \subseteq \mathbf{X} \backslash \{X_i, X_j\}$ by using the G^2 statistic (Spirtes et al., 2000), under the null hypothesis of conditional independency holding. Referring by $N^{ab\mathbf{c}}$ to the number of times that

$X_i = a$, $X_j = b$ and $\mathbf{A} = \mathbf{c}$ simultaneously in the data, G^2 is defined by:

$$G^2 = 2 \sum_{a,b,\mathbf{c}} N^{ab\mathbf{c}} \ln \left(\frac{N^{ab\mathbf{c}} N^{\mathbf{c}}}{N^{a\mathbf{c}} N^{b\mathbf{c}}} \right).$$

The G^2 statistic is asymptotically distributed as χ^2 under the null hypothesis. The χ^2 test returns a p-value, $P_{IT}(X_i, X_j|\mathbf{A})$, that corresponds to the probability of falsely rejecting the null hypothesis given it is true; in MMPC, the effective number of parameters defined in Steck and Jaakkola (2002) is used as degree of freedom. Thus given a significance level α, if $P_{IT} \leq \alpha$ null hypothesis is rejected, that is, X_i and X_j are considered as conditionally dependent. Otherwise, the hypothesis is accepted (abusing somehow of the meaning of the test), and variables are declared conditionally independent. The main idea of MMPC is: given a conditioning set $\mathbf{A}$, instead of considering only $P_{IT}(X_i, X_j|\mathbf{A})$ to decide dependency, it is more robust to consider $\max_{\mathbf{B} \subseteq \mathbf{A}} P_{IT}(X_i, X_j|\mathbf{B})$; that way, the decision is based on more tests; p-values already computed are cached and reused to calculate this maximal p-value. Finally, MMPC build the neighborhood of each variable X_i (called the set of parents and children, or **PC**) by adding successively potential neighbors of X_i from a temporary set $\mathbf{T}$. While conditioning on the actual neighborhood **PC**, the variable $X_k \in \mathbf{T}$ that minimizes the maximal p-value defined before is selected because it is the variable the most related to X_i. During this phase, every variable that appears independent of X_i is not considered anymore and withdrawn from $\mathbf{T}$. Then when X_k is added to **PC**, we test if all neighbors are always conditionally dependent: if some are not, they are withdrawn from **PC** and not considered anymore. This process ends when $\mathbf{T}$ becomes empty.

We present further the details of our implementation of MMPC, referred as Method 2; it is slightly different from the original presentation of MMPC, but the main steps of the algorithm are the same. One can prove by using Theorem 1 that if the independencies are correctly estimated, this Method should return the true skeleton, which should be the case in the sample limit. About computational complexity, one can derive that MMPC should calculate around $O(n^2 2^m)$ tests in average. However, nothing can be said in practice about the maximal size of **PC**, especially if many false dependencies occurs. Therefore, the time complexity of MMPC can be in the worst case $O(n^2 2^n)$.

Method 2 (MMPC). (Tsamardinos et al., 2006)

1: For $\forall X_i \in \mathbf{X}$
2: Initialize $\mathbf{PC} = \emptyset$ and $\mathbf{T} = \mathbf{X} \setminus \{X_i\}$
3: While $\mathbf{T} \neq \emptyset$
4: For $\forall X_j \in \mathbf{T}$, if $\max_{\mathbf{B} \subseteq \mathbf{PC}} P_{IT}(X_i, X_j|\mathbf{B}) > \alpha$ then $\mathbf{T} = \mathbf{T} \setminus \{X_j\}$
5: Define $X_k = \min_{X_j \in \mathbf{T}} \max_{\mathbf{B} \subseteq \mathbf{PC}} P_{IT}(X_i, X_j|\mathbf{B})$ and $\mathbf{PC} = \mathbf{PC} \cup \{X_k\}$
6: For $\forall X_j \in \mathbf{PC} \setminus \{X_k\}$, if $\max_{\mathbf{B} \subseteq \mathbf{PC} \setminus \{X_j\}} P_{IT}(X_i, X_j|\mathbf{B}) > \alpha$ Then $\mathbf{PC} = \mathbf{PC} \setminus \{X_j\}$
7: $\mathbf{N}(X_i) = \mathbf{PC}$
8: For $\forall X_i \in \mathbf{X}$ and $\forall X_j \in \mathbf{N}(X_i)$
9: If $X_i \notin \mathbf{N}(X_j)$ Then $\mathbf{N}(X_i) = \mathbf{N}(X_i) \setminus \{X_j\}$

5.2 Learning a Super-structure with MMPC

To emphasize the feasibility of learning a super-structure from data, we study how changes the skeleton learned by MMPC while α increases, considering various cases of $(n, \tilde{m}, d)$. As proposed

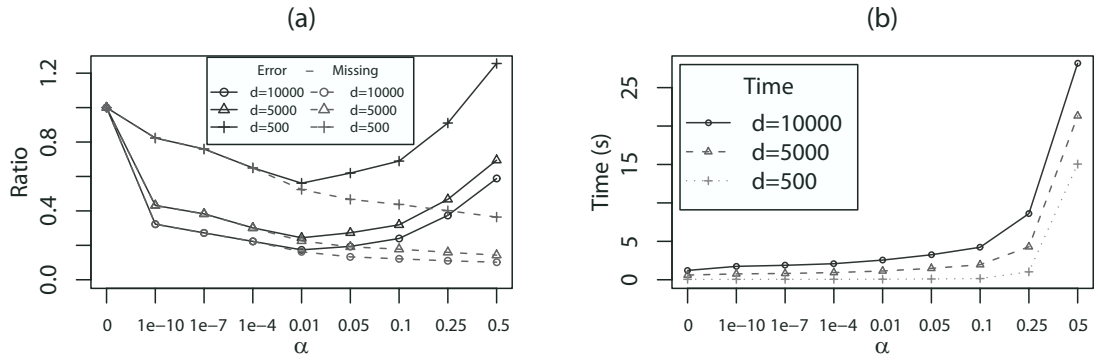


Figure 6: Effects of d and α on the results of Method 2.

before, we average our criteria of interest over $g = 50$ different graphs for every set of parameters. In the present case our criteria are: the time of execution *Time*, the ratio of wrong edges (missing and extra) $Error$, and the ratio of missing edges *Miss* of the learned skeleton. Here again, these ratios are defined while comparing to the true skeleton and dividing by its total number of edges $\lfloor \frac{n\tilde{m}}{2} \rfloor$. While learning a skeleton, $Error$ should be minimized; however in the case of learning a super-structure, *Miss* is the criterion to minimize.

Results 3 $\lim_{\alpha \to 1} Miss(\alpha) = 0$, *which validates our proposition of using higher* α *to learn super-structures (cf. Figure 6(a)). Of course, we obtain the same results with increasing data,* $\lim_{d \to \infty} Miss(\alpha) = 0$. *However, since when* $\alpha \to 1$, $Time(\alpha) \approx O(n^2 2^{n-2})$, *high values of* α *can be practically infeasible (cf. Figure 6(b)). Therefore, to escape a time-consuming structure-learning phase,* α *should be kept under* 0.25 *if using MMPC.*

In Figure 6(a), one can also notice that $Error$ is minimized for $\alpha \approx 0.01$, that is why such values are used while learning a skeleton. Next, we summarize the effect of n and $\tilde{m}$ on the criteria:

Results 4 *Increasing* α *improves uniformly the ration of missing edges independently of n and* $\tilde{m}$ *(cf. Figure 7(c) and (d)).* $Miss(\alpha)$ *is not strongly affected by increasing n but it is by increasing* $\tilde{m}$*; thus for dense graphs, the super-structures approximated by MMPC will probably not be sound.*

Previous statement could be explained by the fact that when $\tilde{m}$ increases, the size of conditional probability tables of each node increases enormously. Thus, the probability of creating weak or nearly unfaithful dependencies between a node and some of its parents also increases. Therefore, the proportion of edges that are difficult to learn increases as well. To complete analysis of Figure 7, we can notice as expected that *Time* increases on a polynomial manner with n (cf. Figure 7 (e)), which penalizes especially the usage of high α. $Error(\alpha)$ is minimized in general for $\alpha = 0.01$ or $\alpha = 0.05$ depending on n and $\tilde{m}$: for a given $\tilde{m}$, lower α (such as $\alpha = 0.01$) are better choices when n increases (cf. Figure 7(a)); conversely, if $\tilde{m}$ increases for a fixed n, higher α (such as $\alpha = 0.05$) are favored (Figure 7(b)). This can be justified, since if $\tilde{m}$ is relatively high, higher α that are more permissive in terms of dependencies will miss less edges, as opposed to lower ones with finite set of data.

In conclusion, although $\alpha \approx 0.01$ is preferable while learning a skeleton with MMPC, higher significance leves can be used to reduce the number of missing edges, and approximate a super-structure. Still, in this case, due to the exponential time complexity of MMPC if α is too high,

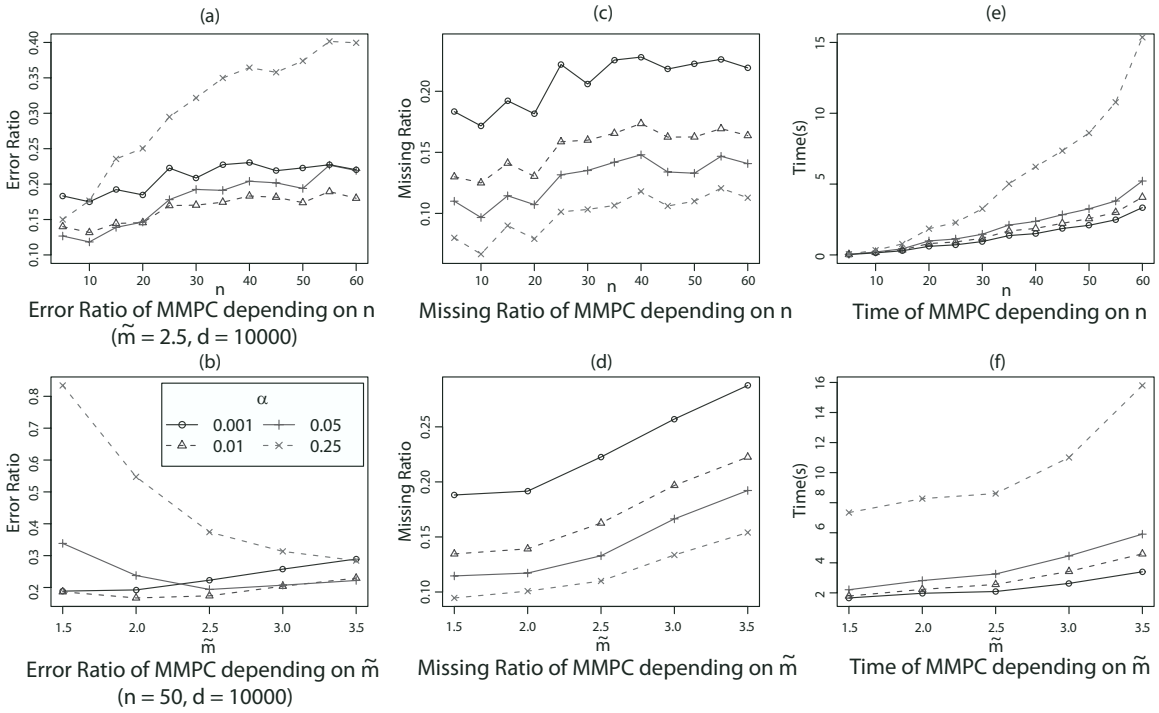


Figure 7: The criteria depending on $\tilde{m}$ and n, for α in $[0.001, 0.01, 0.05, 0.25]$.

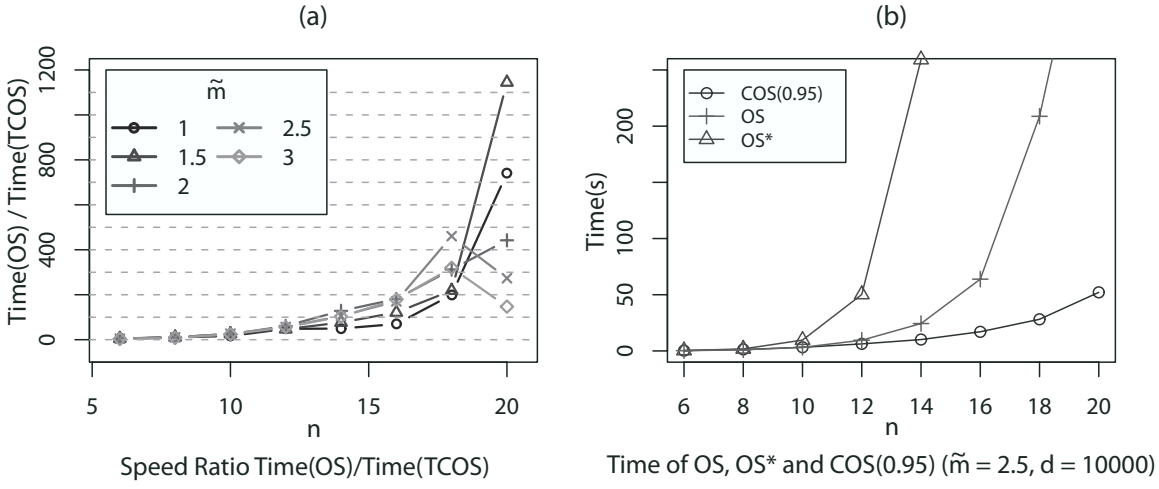


Figure 8: Comparing time complexity of OS, TCOS, and COS(α)

values of α should be selected to compromise between time complexity and ratio of missing edges. Except for $n < 30$ with $\tilde{m} < 2.5$ where $\alpha = 0.25$ is feasible and gives good results, the learned super-structure will be probably incomplete, especially if the original graph is dense: consequently, COS will not perform as well as OS when S is approximated with MMPC.

5.3 Comparison of OS and COS

In this second series of experiment, we compare Algorithm 1 and Algorithm 2 over a large sampling of graphs, for confirming and evaluating results presented in Table 1. Algorithm 1, referred as OS, is always given the true maximal number of parents c for each structure learning: this way, its execution time is considerably reduced, and we could conduct our experiments in a reasonable time. Still, to emphasize that this prior knowledge is considerably improving the speed of OS, we also considered another version of Algorithm 1 that uses in all cases a standard maximal number of parents equal to 10: it is referred as OS*. Regarding Algorithm 2, two cases are took into consideration:

- TCOS: a sound super-structure is given to the algorithm; we use the true skeleton.
- COS(α): S is learned from the data by using MMPC and a significance level α: a wide range of values are tested.

However, we know from previous section that MMPC will probably learn an incomplete super-structure, and that $G_{\mathrm{COS}(\alpha)}$ will be penalized both in its score and accuracy. Therefore, we check the effect of applying a post-processing unconstrained hill climbing search starting from $G_{\mathrm{COS}(\alpha)}$. In fact, $G_{\mathrm{COS}(\alpha)}$ might not be a local optimum if the super-structure constraint is removed; further, it could be a good starting point for a hill climbing search. The post processed version of COS is referred as COS+. Finally we compare all those algorithms for every $n \in [6,8,\cdots,20]$, $\tilde{m} \in [1,1.5,\cdots,3]$, $d \in [500,1000,\cdots,10000]$ while averaging the criteria of interest over $g = 30$ graphs: in total 4800 random graphs were used. Since there was no relevant differences depending on d, only the results for $d = 10000$ are reported here.

Results 5 *As expected, TCOS and COS(α) proceed exponentially faster than OS, even with $\alpha \approx 1$ (cf. Figures 8 (a) and (b)).*

Interestingly, the speed factor in Figure 8(a), is not purely exponential, especially for higher $\tilde{m}$. This is because, the complexity of OS is due to the costly $O(n^c)$ score calculations (here $c = O(\tilde{m})$) and the $O(2^n)$ steps (b) and (d) that dominate the complexity only for large n. As for TCOS, it only calculates $O(n)$ scores and needs around $O(\delta_{\tilde{m}}^n)$ steps. Thus, the speed factor starts by increasing fast with n because of the $O(n^c)$ scores of OS, before decreasing to behave asymptotically as $O(\left(\frac{2}{\delta_{\tilde{m}}}\right)^n)$. In Figure 8(b), the complexity of OS* is also represented: without limitation on the number of parents, the speed ratio would be purely exponential.

To evaluate the quality of the graphs learned depending on the algorithm used, for $n = 12$ and $\tilde{m} = 2$ we compare the SER and the score ratio of $G_{\mathrm{COS}(\alpha)}$ and $G_{\mathrm{COS}(\alpha)+}$ depending on α with the ones of G_{OS} and G_{TCOS} in Figure 9. Then in Figure 10, the criteria are represented depending on n (with $\tilde{m} = 2.5$), and on $\tilde{m}$ (with $n = 16$) for every algorithms: here, just three different values of α are considered (0.001, 0.05 and 0.75).

Results 6 *In average, G_{TCOS} has a slightly lower score than G_{OS} (cf. Figure 9 (b)), but it is more accurate (cf. Figure 9 (c), Figures 10 (c) and (d)): it implies that global optima contain in general extra edges. Hence, Algorithm 2 is preferable if a sound constraint is known.*

This important result emphasizes again that the optima of a score function with finite data are not the true networks usually: some false edges improve the score of a graph. Therefore, struc-

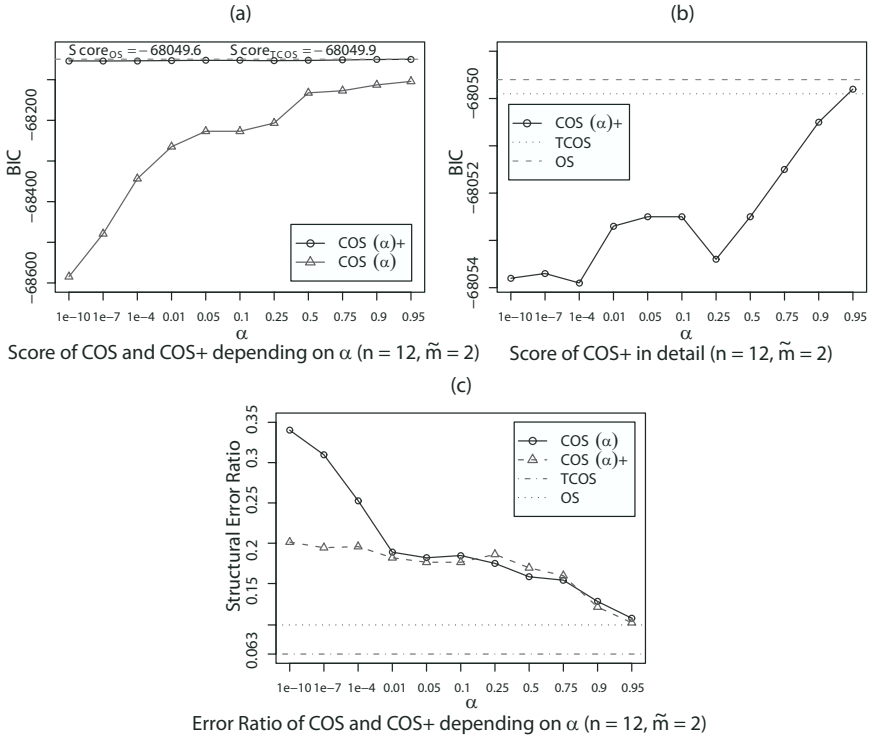


Figure 9: Score and SER for COS(α) and COS(α)+

tural constraint should be generalized whenever a sound super-structure is known: by doing so, the resulting graphs although having a lower score can be more accurate. It is the case for TCOS.

Results 7 *Although* $Score(G_{\mathrm{COS}(\alpha)})$ *converges to* $Score(G_{\mathrm{OS}})$ *when* α *increases (cf. Figure 9(a)), it is usually far lower, and worsen with larger networks or denser networks (cf. Figures 10 (a) and (b)). However,* $SER(G_{\mathrm{COS}(\alpha)})$ *is converging faster to* $SER(G_{\mathrm{OS}})$ *when* α *increases, which enables to find relatively accurate results for* $\alpha \geq 0.01$ *(cf. Figure 9(c)). In addition, n does not affect sensitively* $SER(G_{\mathrm{COS}(\alpha)})$ *(cf. Figure 10 (c)), neither does* $\tilde{m}$ *if* α *is enough high, that is,* $\alpha \geq 0.05$ *(cf. Figure 10 (d)).*

At first sight, these results sound really negative with regard to COS algorithm, or more exactly, with regard to the super-structure learned by MMPC. However, although $Score(G_{\mathrm{COS}(\alpha)})$ is disappointingly low, $SER(G_{\mathrm{COS}(\alpha)})$ is still relatively good: while $SER(G_{\mathrm{OS}}) \approx 0.1$, $SER(G_{\mathrm{COS}}) \approx 0.2$ for α as small as 0.01 in Figure 9 (c). Of course, following Results 6, we could think that for enough high α, $SER(G_{\mathrm{COS}(\alpha)}) \leq SER(G_{\mathrm{OS}})$, but it seems to be hardly feasible while using MMPC: in this case, when IT are relaxed (i.e., $\alpha \to 1$), false edges that improve the score are learned faster than the true ones missing. Still, as it appears in Figures 10 (c) and (d), for $\alpha = 0.75$, COS finds graphs nearly as accurate as the optimal ones. Therefore, when OS is not feasible, COS could be an alternative to greedy searches (such situation is studied in following Sections). Interestingly, our assumption about the potential interest of a hill climbing starting from $G_{\mathrm{COS}(\alpha)}$ are encouraged since $G_{\mathrm{COS}(\alpha)}$ has a low score while being relatively accurate. Next, we focus on COS+.

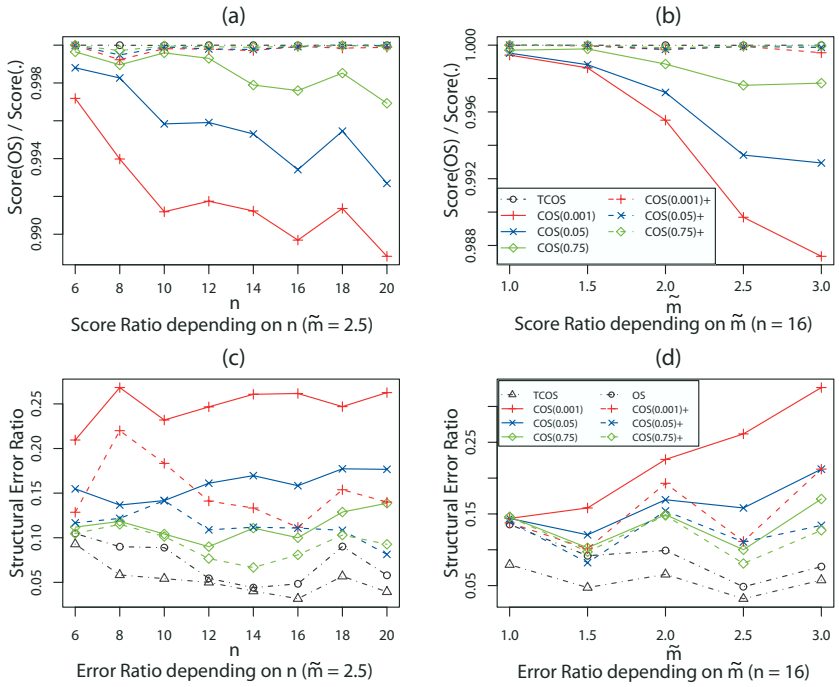


Figure 10: Evolution of Score and SER for every algorithm depending on n and $\tilde{m}$.

Results 8 *For any* α*, after post-processing Score*$(G_{\mathrm{COS}(\alpha)+})$ *is nearly as good as Score*(G_{OS}) *(cf. Figure 9 (a), Figures 10 (a) and (b)).* $G_{\mathrm{COS}(\alpha)+}$ *is in general more accurate than* $G_{\mathrm{COS}(\alpha)}$ *(cf. Figures 10 (c) and (d)), but it is not always the case. Still, SER is clearly improved for* $\alpha < 0.01$ *(cf. Figure 9 (c)): COS+ could be feasible on larger networks while giving interesting results if sufficiently small* α *are used.*

As expected, $G_{\mathrm{COS}(\alpha)}$ is not a local optima, and is at least a good starting point to maximize the score. Figure 9 (b) presents a detailed view of *Score*$(G_{\mathrm{COS}(\alpha)+})$: with post processing, we cannot be sure that *Score*$(G_{\mathrm{COS}(\alpha)+})$ increases strictly with α because of the greedy nature of hill climbing. For the same reasons, previous results concerning the SER are true only in average. To complete Results 8, we should remark that post-processing also appears to improve SER for dense graphs (cf. Figure 10 (d)). This is probably due to the fact that MMPC misses many true edges when the structure is dense: the greedy search would add more true edges missing than false ones, improving consequently the SER.

To summarize this section, TCOS is superior to OS in terms of structure accuracy, while performing faster. However, when S is learned from data, COS cannot perform as well as OS. One efficient strategy to improve both score and accuracy of G_{COS} is to proceed to a post-processing unconstrained hill climbing search. With such an improvement, both score and accuracy of $G_{\mathrm{COS}(\alpha)+}$ become similar for any α used, despite a relative superiority for higher significance levels. Consequently, it would be interesting to compare COS and COS+ to greedy searches, to see if they enable to learn efficiently more accurate graphs than other methods, while being given incomplete super-structures.

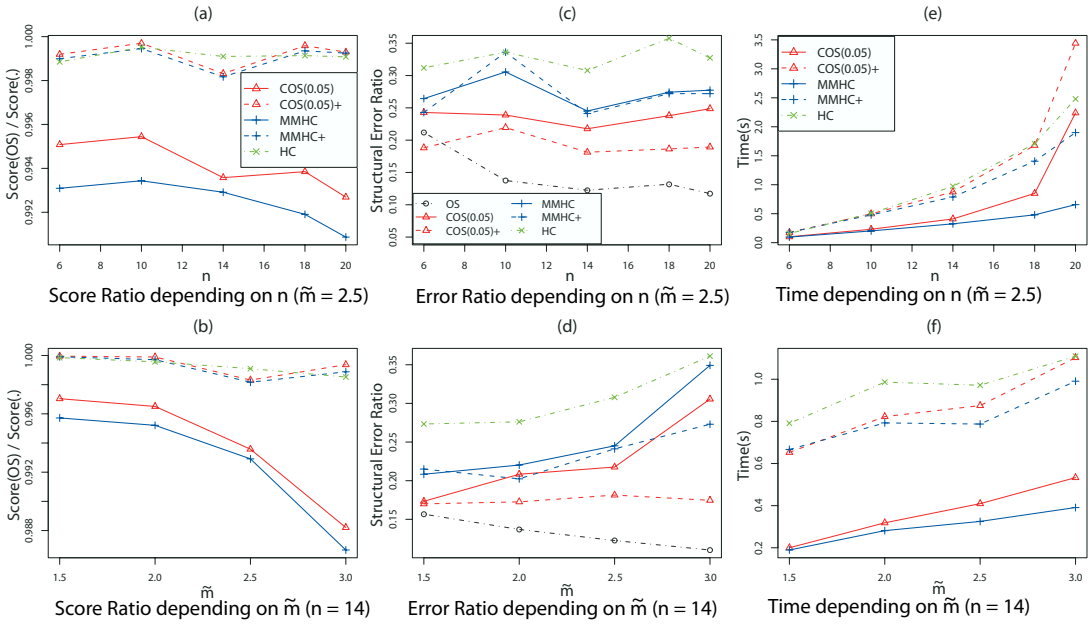


Figure 11: Score, SER and time of every algorithm depending on n and $\tilde{m}$.

5.4 COS and COS+ Compared to Heuristic Searches

In this last series of experiment, we compare COS and COS+ to other greedy searches. The structural constraint is learned with MMPC and $\alpha = 0.05$, it is used by COS and also by a constrained hill climbing search (MMHC). Like COS+, G_{MMHC} is used to start an unconstrained hill climbing (MMHC+). Finally, a classic hill climbing search from the empty graph is also considered (HC). Actually, OS is also performed, but it is not directly referred in this section. Every $n \in [6,10,14,18,20]$, $\tilde{m} \in [1,1.5,2,2.5,3]$, $d \in [500,1000,5000,10000]$ are considered and criteria are averaged over $g = 30$ graphs: in total 3000 graphs are used.

Figure 11 presents the three criteria in function of n and $\tilde{m}$; here $d = 5000$. Synoptic results are presented in Figures 12 and 13, and summarized in Tables 3 and 4. To obtain these tables, we ordered algorithms by score for all the 100 triplets $(n,\tilde{m},d)$ (the scores after averaging over the g graphs were used). Subsequently, we counted the number of times each algorithm was at a given rank: If two algorithms had equal results, they were given the same rank (cf. Table 3). We then performed a similar ranking by comparing SER; the first ranked algorithm being the one that minimized SER (cf. Table 4). In Figures 12 and 13 ranks are represented by colors and all algorithms can be directly compared for every triplet $(n,\tilde{m},d)$. However, while considering the impact of these results, one should take the fact that criteria are compared after averaging into account since it accentuates the contrast between algorithms.

Results 9 *In general,* $Score(G_{\text{COS+}}) \geq Score(G_{\text{MMHC+}}) \geq Score(G_{\text{HC}}) \geq Score(G_{\text{COS}}) \geq Score(G_{\text{MMHC}})$ *(cf. Figures 11 (a), (b), 12 and Table 3). In other words, the super-structure penalizes the score in comparison with HC; however, starting a greedy search from a constrained optimal graph enables to find better scores than HC.*

Many of these inequalities are naturally following the definition of the algorithms; another part comes from the fact that S is incomplete and penalizes the score of the results. Moreover, our

Algorithm	1st	2nd	3rd	4th	5th
COS+	84	8	8	0	0
MMHC+	12	70	18	0	0
HC	7	28	65	0	0
COS	0	1	8	91	0
MMHC	0	0	1	8	91

Table 3: Classification by score.

Algorithm	1st	2nd	3rd	4th	5th
COS+	85	9	6	0	0
COS	14	52	22	12	0
MMHC+	6	29	28	35	2
MMHC	0	12	34	31	23
HC	1	2	17	24	56

Table 4: Classification by SER.

experiments confirm that in nearly every case $Score(G_{\text{COS+}}) > Score(G_{\text{MMHC+}}) > Score(G_{\text{HC}})$; A reason could be that both constrained algorithms find good constrained optima, which are not locally optimal when withdrawing the constraint. Then, since G_{COS} is better than G_{MMHC}, it leads to better local optima when applying the HC post-processing.

Results 10 *In general,* $SER(G_{\text{COS+}}) \leq SER(G_{\text{COS}}) \leq SER(G_{\text{MMHC+}}) \leq SER(G_{\text{MMHC}}) \leq SER(G_{\text{HC}})$ *(cf. Figures 11 (c), (d), 13 and Table 4). The superiority of COS and COS+ is particularly clear with larger data sets (cf. Figure 13).*

Previous results are experimentally more confused than Results 9 since there are no theoretical evidences about SER while comparing algorithms, especially as regards to MMHC+ and MMHC. However, COS+ is clearly demonstrated to be in general the most accurate search. Interestingly, COS+ can be viewed as a bridge between OS (when $\alpha = 1$) and HC (when $\alpha = 0$). Of course, as long as OS is feasible, and without a sound constraint, COS+ is not really needed. However, it is of a certain interest for $n > 30$. Its superiority as compared to HC comes from the fact that some parent sets are optimally learned by COS even with the constraint. Such parent sets could not be learned by the greedy strategy of HC in any cases: as for example in the case of a XOR structure or any configurations for which HC should add several parents in the same time to obtain a score improvement. That way, a HC starting from G_{COS} would benefit from such correct parent sets and find more accurate graphs than a HC starting from the empty graph.

Although for the small networks considered here ($n \leq 20$) our algorithms are as fast as other greedy approaches (cf. Figures 11 (e) and (f)), of course their exponential complexity keeps us from applying them to really large networks without decreasing α. Therefore, COS is restricted to be used only for small networks or sparse ones (Figure 5 (d) gives an idea of what kind of graphs can be considered). COS+ could be used for larger graphs by using lower α as we will see in the next experiment.

5.5 A Real Case: The ALARM Network

In this last experiment, we illustrate a practical usage of COS and COS+ by studying a well-known Bayesian network example, the ALARM network (Beinlich et al., 1989). This graph has $n = 37$ discrete variables having from 2 to 4 states: it is too large to be learned by OS. The maximal in-degree is $c = 4$, the maximal neighborhood is $m = 6$, and the structure is relatively dense since $\tilde{m} \approx 2.5$. Incidentally, the true skeleton entails $86818458 \approx 2^{26}$ connected subsets: if it was given as a prior knowledge, Algorithm 2 could be applied. However, as in previous experiments, S is learned by using MMPC. Nevertheless, α should be enough small so that the super-structure is sparse enough to let execute COS.

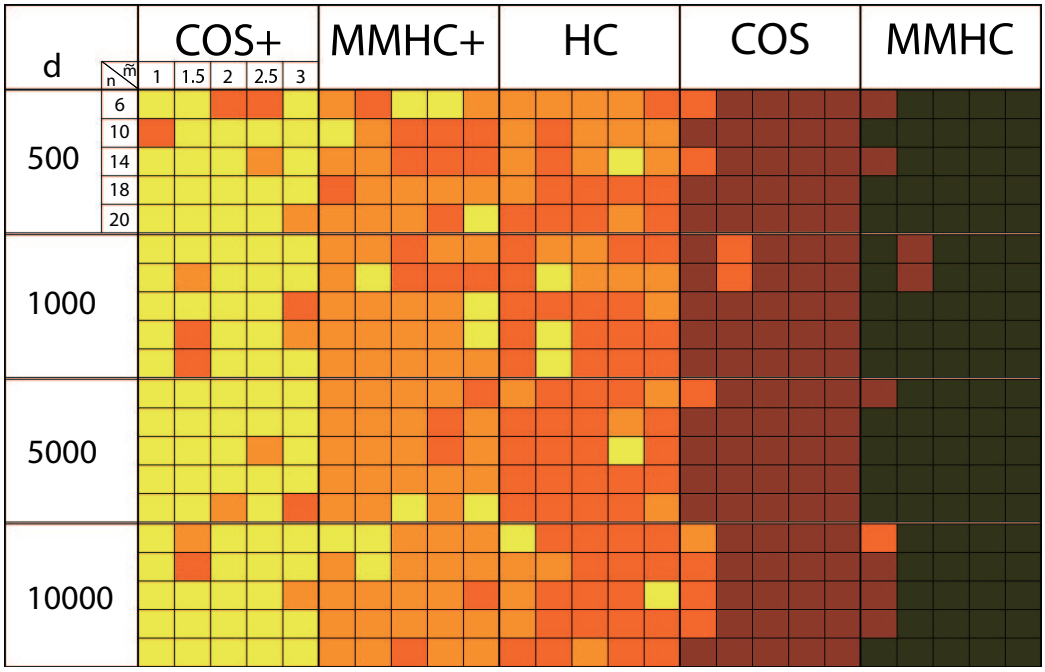


Figure 12: Comparison by score: for every $(n, \tilde{m}, d)$, the worse is the rank of an algorithm (i.e., the lower is the score), the darker is the box.

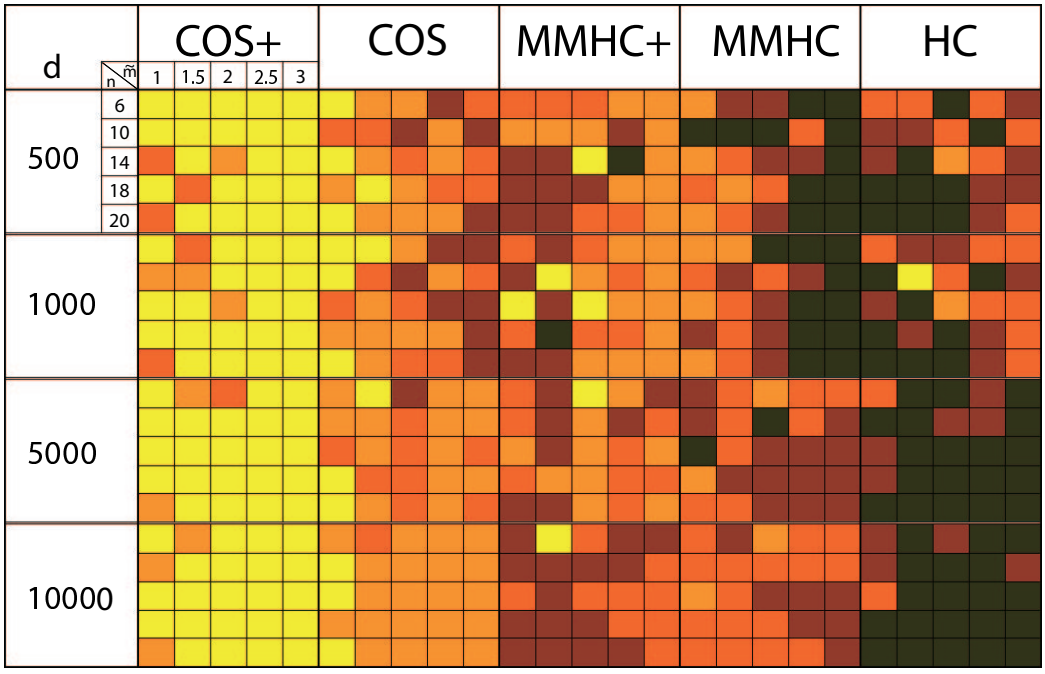


Figure 13: Comparison by SER: for every $(n, \tilde{m}, d)$, the worse is the rank of an algorithm (i.e., the higher is the SER), the darker is the box.

		COS	COS+	MMHC	MMHC+	HC
$d = 3000$	BIC	-33340.5	-32849.5	-33492	-33116.1	-33299.3
$\alpha = 10^{-4}$	SER	0.15	0.12	0.29	0.38	0.45
$t = 24$	Time(s)	4.6	7.1	3.6	6.0	5.0
$d = 3000$	BIC	-33680.8	-32821.4	-33653.2	-33133.2	-33349.9
$\alpha = 10^{-7}$	SER	0.17	0.10	0.29	0.39	0.47
$t = 4$	Time(s)	3.5	6.0	2.7	5.3	5.1
$d = 10000$	BIC	-107834	-106246	-108035	-106808	-107383
$\alpha = 10^{-7}$	SER	0.13	0.08	0.26	0.37	0.51
$t = 270$	Time(s)	10.5	19.2	14.7	23.4	18.8
$d = 10000$	BIC	-108753	-106468	-108420	-107066	-107571
$\alpha = 10^{-10}$	SER	0.15	0.10	0.25	0.35	0.49
$t = 72$	Time(s)	9.1	17.7	12.8	21.4	18.0

Table 5: Averaged criteria for some pairs (d, α) considered.

We consider the same algorithms than in Section 5.4, apart from OS. The significance level used to learn the skeleton of MMHC is always set to 0.01, while various small values of α are tested when learning the super-structure used by COS. For every $d \in [3000, 5000, 10000]$, we start by learning S with MMPC(α) to run COS afterwards. Since, the execution time of COS can be huge depending on S, COS is stopped if its execution exceeds 20 seconds (on a cluster of 96 CPUs of 1050 MHz each, with a total memory of 288GB). In that case, the experience is restarted on a new set of data. If COS finishes on time, all other algorithms are also applied. Previous steps are repeated 20 times for every pair (d, α). We also count the number t of times that COS had to be restarted. Finally, the criteria are averaged over the 20 learned graphs as in the previous Sections.

In all cases results were sensibly the same: Table 5 presents the results obtained with the minimal and maximal values of d and α considered. With larger data sets, smaller significant levels are required in order to obtain an enough sparse S since true edges are learned more confidently. This is why, we can observe that t decreases with smaller α and increases with larger d (cf. Table 5). Therefore, for $d = 10000$ we tried values of $\alpha \in [10^{-7}, \cdots, 10^{-10}]$, while for other data sets $\alpha \in [10^{-4}, \cdots, 10^{-7}]$. In Figures 14 (a) and (b), box plot is used to layout the SER and score of every algorithm over the 20 repetitions of structure learning for $d = 5000$ and $\alpha = 10^{-7}$. Figures 14 (c) and (d) represent the SER of COS and COS+ depending on d and α, respectively.

Results 11 *COS+ is the best algorithm in terms of score, followed by MMHC+ (cf. Table 4 and Figure 14 (a)), while COS+ and COS learn the most accurate networks (cf. Table 4 and Figure 14 (b)). Accuracy of these algorithms is not markedly modified by smaller α (cf. Figure (d)) and slightly improves with bigger data sets (cf. Figure 14 (c)).*

In other words, the present results are in total agreement with the ones derived from smaller and random networks in the previous section. Actually, with ALARM network the contrast between algorithms is even clearer. This is probably because random networks are harder to learn than real ones since the parameters randomly selected can entail nearly unfaithful probability distributions. Interestingly, even with the smallest α used (i.e., $\alpha = 10^{-10}$ in the case of $d = 10000$), only around

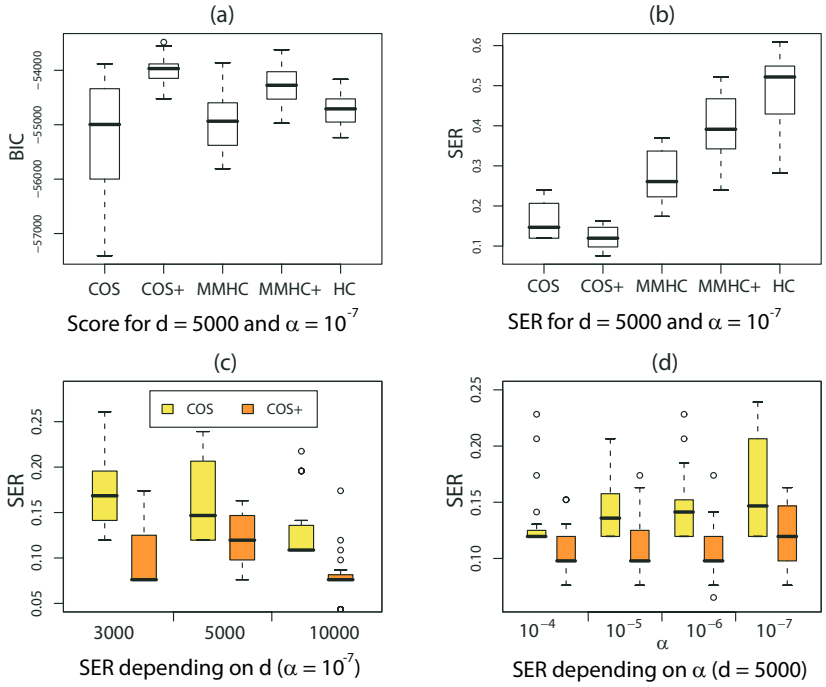


Figure 14: Some detailed results on ALARM network.

6 edges were missing to S in order to be sound. Besides, although COS ran in few seconds for some S, one extra edge in the same super-structure could lead to drastically longer computations. Such a problem is easily understood in terms of the number of connected subsets and underline the fact that some edges contribute particularly to the increase of $|Con(S)|$. Therefore it could be interesting to develop a method to select and withdraw some edges from a learned super-structure, to enable COS+ for larger networks: this way, extremely small α would not be required anymore.

6. Discussion and Conclusion

To conclude our discussion, we would like to recall and summarize the main results of our actual research. First, it is possible to reduce the complexity of an optimal search of an exponential factor by using a structural constraint such as a super-structure. It is then possible to consider larger networks having a sparse skeleton. Moreover, if this super-structure is sound, the accuracy of the resulting graph is improved. Consequently, more attention should be paid to learning sound super-structures rather than true skeleton from data. This should be an easier task and it might improve both speed and accuracy of other search strategies as well, except greedy HC. Next, we outline bellow some strategies that could benefit from a sound super-structure.

In addition, current IT methods that learn a skeleton can be used for approximating a super-structure by relaxing the independency testing. However, as revealed our experiments with MMPC, sound super-structures are rarely learned except for high values of α that implies denser structures and especially a long computation. Although some other IT approaches, such as the randomized version of GS (Margaritis and Thrun, 2000), could solve the problem of complexity, they would probably face the same difficulty to learn efficiently at least every true edge. Indeed, they were de-

signed from the viewpoint of learning the true skeleton, and thereby they should also reject potential extra edges. Learning sound and sparse super-structures is a problem that requires to be considered as a whole, which is not our main concern in the present paper. Therefore, to offset the incompleteness of the super-structures learned with MMPC that weakens the results of our algorithm COS, we developed a greedy post processed search COS+. This algorithm enables to balance between speed and accuracy as it sets up a bridge between optimal search and hill climbing search. In practice, COS+ is demonstrated with success on the ALARM network. It is theoretically feasible for graphs of any sizes but leads to a problem for selecting the significance level; further, we expect that the larger is the graph, the less COS+ should improve over HC.

Therefore, our future research will concentrate on an elaboration of new greedy strategies that benefit from super-structure constraint. As shown in Section 4.2, if S is not having a very high maximal degree (i.e., $m < 20$), F can be calculated even for large networks. Then, by using formula (3) of Section 3.2 we can build in linear time the best constrained graph of a given ordering. This way a fast greedy search over topological orderings is efficiently feasible. With a good set of greedy operations, a constrained optimal graph could be quickly approached using a hill climbing over orderings, without having to calculate M values. Moreover, even for higher m, by fixing a limited number of parents c, we could also manage to calculate F. Consequently, such ordering-based strategy is theoretically feasible for any network size and only need to be evaluated experimentally.

Our second idea concerns only constrained optimal searches. When looking for an optimal graph on $\mathbf{A} \in Con(S)$, if there exists $X_i \in \mathbf{A}$ such that $\mathbf{A}\backslash\{X_i\}$ is unconnected, an optimal graph can be found on another manner. We just need to consider every candidate parent set on the neighborhood of X_i, and search for each of them separately an optimal graph on each connected component of $\mathbf{A}\backslash\{X_i\}$. Thus, if S is a tree, it would be feasible to find an optimal graph in polynomial time with this different strategy. Then, we could develop an algorithm that would change S into a tree, learn an optimal graph that would be post-processed greedily to add missing edges.

Besides, as illustrated in Section 4.4, COS can be used for large graphs, if S is sufficiently sparse. Alternatively, when learning a super-structure, a "score" could be given to every edge that would evaluate the "strength" of the dependency represented. For example, the highest p-value encountered while learning the super-structure with an IT approach could be used. Thus, COS could be used sequentially: the subset of the strongest edges would be considered first, while learning a temporary optimal graph that would be used as a prior knowledge for a second search, this time optimally "adding" a second set of edges. Therefore, by considering successively all the edges of S by a sufficient number of sparse layers, we might be able to approximate accurately an optimal graph. This strategy is potentially interesting since it always assumes the graph as a whole, although the layer of edges should be mainly unconnected to allow COS to be applied to large networks. However, other strategies such as rebuilding optimally the structure locally before merging results could also be developed.

Finally if any of these algorithms were giving convincing empirical proofs of their capacities to learn accurately large networks, we also would like to design strategies for learning sparse and sound super-structures.

Acknowledgments

We would like to thank the reviewers and the associate editor who helped us considerably to improve the quality of this paper, and gave us many valuable suggestions. We should also thank the Super Computer System, Human Genome Center, Institute of Medical Science, University of Tokyo, for letting us use their super computer with which we realized our experiments. We also acknowledge the Japanese Ministry of Education, Culture, Sports, Science and Technology for its financial support.

Appendix A.

Proof of Proposition 1. The complete proof we found is long, complex, and not of the greatest theoretical interest. Hence we will only present the main steps of the demonstration here without explicitly developing every step.

- We start by considering any binary tree, which can be generated by successively applying the following operation to the tree reduced to one leaf: Select a leaf, and change it to a node with two leaves. Then, for the set of trees that have been generated by applying this transformation t times, we define $U(t)$ as the maximum number of sub-trees in these trees, and $V(t)$ as the maximum number of sub-trees that include a given leaf. Given such a tree T, we apply the transformation to a leaf L_i. This generates a node N_i' and two leaves L_{i1}' and L_{i2}', where the prime indicates that we refer to the elements of the newly generated tree T'. By calling $Sub(L)$ the set of sub-trees that contain L, we have: since $|Sub(L_i)| \leq V(t)$, $|Sub(L_{i1}')| \leq 1+2V(t)$. If L_i had a brother L_j, we can show that $|Sub(L_j')| \leq \frac{5}{2}V(t)$ since only half the sub-trees of $Sub(L_j)$ contain L_i. Then, for leaves L_k at agreater distance from L_i than L_j, since the proportion of sub-trees of $Sub(L_k)$ that also contain L_i is decreasing, $Sub(L_k')$ is less increased. Thus, we can conclude that $V(t+1) < \frac{5}{2}V(t)$, and that $V(t) < U(t) < O((\frac{5}{2})^t)$. Finally, since $n = 2t-1$, we derive that $|Con(S)| < O({\alpha_3}^n)$ with $\alpha_3 = \sqrt{\frac{5}{2}}$.
- Hereafter, we can apply the same reasoning to any k-tree, and find that $|Con(S)| < O({\alpha_{k+1}}^n)$ with $\alpha_{k+1} = (\frac{2^k+1}{2})^{\frac{1}{k}}$.
- Then, start the fastidious part of the demonstration by considering a connected forest S of maximal degree m. The idea of the demonstration is to first show that when we consider two nodes of the tree, we can transfer from one node to the other every sub-rooted trees while only increasing the number of connected subsets (actually it is possible to express exactly the variation in the number of connected subsets depending on the unique path between the two nodes and sub-rooted trees of the nodes in this path). Then, by selecting one of the nodes of degree m from S, and fixing it as the root R_0, we can sequentially build a nearly m-1-tree S', except for the root, and maybe for one node R_i, while increasing the number of connected subsets. We do this by considering depth by depth descendants of R_0 and transferring sub-rooted trees from one to the other until all of them at a given depth have $m-1$ or 0 children, taking sub-rooted trees deeper if necessary. Then, if the sons of R_i are not all leaves, we can continue to apply transformations between it and its descendants until obtaining a nearly m-1-tree S" except for its root, and one of its nodes that contain only p leaves. Since $p \leq m-1$, we add $m-p$ leaves to this node, and we select as the root of this final tree S_f one of the

leaves; S_f is like a m-1-tree except for the root that has only one son. In such a case, we can use the results found for such structures, neither the root will sensitively affect the number of connected subsets, nor the nodes that we added at the end; because they can be put in the constant of $O(\alpha_m{}^n)$, which is an upper bound since we built S_f always taking care to increase the number of connected subsets. ■

References

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

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

A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The travelling salesman problem in bounded degree graphs. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, 2008.

R. Bouckaert. *Bayesian Belief Networks from Construction to Inference*. PhD thesis, University of Utrecht, 1995.

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

D. Chickering. Learning Bayesian networks is NP-complete. *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130, 1996.

D.M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 87–98. Morgan Kaufman, 1995.

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

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

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

R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

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

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

C.N. Glymour. *The Mind's Arrows: Bayes Nets & Graphical Causal Models in Psychology*. MIT Press, 2001.

C.N. Glymour and G.F. Cooper. *Computation, Causation, and Discovery*. AAAI Press / The MIT Press, 1999.

D. Heckerman. A tutorial on learning with Bayesian networks. Technical report, Microsoft Research, 1996.

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

S. Imoto, T. Goto, and S. Miyano. Estimation of genetic networks and functional structures between genes by using Bayesian networks and nonparametric regression. *Pacific Symposium on Biocomputing*, 7:175–186, 2002.

M. Kalisch and P. Bühlmann. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In S.A. Solla, T.K. Leen, and K.R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 505–511. MIT Press, 2000.

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

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

R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

S. Ott and S. Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14:124–133, 2003.

S. Ott, S. Imoto, and S. Miyano. Finding optimal models for small gene networks. *Pacific Symposium on Biocomputing*, 9:557–567, 2004.

S. Ott, A. Hansen, S.Y. Kim, and S. Miyano. Superiority of network motifs over optimal networks and an application to the revelation of gene network evolution. *Bioinformatics*, 21(2):227–238, 2005.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman Publishers, San Mateo, CA, 1988.

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

R. Robinson. Counting labelled acyclic digraphs. In *New Directions in the Theory of Graphs*, pages 239–273. Academic Press, 1973.

G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.

E. Segal, D. Pe'er, A. Regev, D. Koller, and N. Friedman. Learning module networks. *Journal of Machine Learning Research*, 6:557–588, 2005.

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Conference on Uncertainty in Artificial Intelligence*, pages 445–452, 2006.

A.P. Singh and A.W. Moore. Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University, 2005.

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

H. Steck and T. Jaakkola. On the Dirichlet prior and Bayesian regularization. In *Advances in Neural Information Processing Systems*, volume 15, 2002.

J. Suzuki. Learning Bayesian belief networks based on the MDL principle: an efficient algorithm using branch and bound technique. *IEICE Transactions on Information and Systems*, 12, 1998.

M. Teyssier and D. Koller. Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence*, 2005.

I. Tsamardinos, L.E. Brown, and C.F. Aliferi. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 2006.