Learning Bayesian networks by hill climbing: Efficient methods based on progressive restriction of the neighborhood

Article *in* Data Mining and Knowledge Discovery · May 2011

DOI: 10.1007/s10618-010-0178-6 · Source: DBLP

CITATIONS
94

READS
2,412

3 authors:

José A. Gámez
University of Castilla-La Mancha
182 PUBLICATIONS **1,936** CITATIONS
SEE PROFILE

Juan Luis Mateo
University of Oviedo
49 PUBLICATIONS **913** CITATIONS
SEE PROFILE

Jose Miguel Puerta
University of Castilla-La Mancha, Albacete, Spain
117 PUBLICATIONS **1,456** CITATIONS
SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

Learning Bayesian networks by hill climbing: Efficient methods based on progressive restriction of the neighborhood

José A. Gámez · Juan L. Mateo · José M. Puerta

Apcepted: 11/05/2010 Published: 01/01/2011
The final publication is available at Springer via http://dx.doi.org/10.1007/s10618-010-0178-6

Abstract Learning Bayesian networks is known to be an NP-hard problem and that is the reason why the application of a heuristic search has proven advantageous in many domains. This learning approach is computationally efficient and, even though it does not guarantee an optimal result, many previous studies have shown that it obtains very good solutions. Hill climbing algorithms are particularly popular because of their good trade-off between computational demands and the quality of the models learned. In spite of this efficiency, when it comes to dealing with high-dimensional datasets, these algorithms can be improved upon, and this is the goal of this paper. Thus, we present an approach to improve hill climbing algorithms based on dynamically restricting the candidate solutions to be evaluated during the search process. This proposal, dynamic restriction, is new because other studies available in the literature about restricted search in the literature are based on two stages rather than only one as it is presented here.

In addition to the aforementioned advantages of hill climbing algorithms, we show that under certain conditions the model they return is a minimal I-map of the joint probability distribution underlying the training data, which is a nice theoretical property with practical implications. In this paper we provided theoretical results that guarantee that, under these same conditions, the proposed algorithms also output a minimal I-map. Furthermore, we experimentally test the proposed algorithms over a set of different domains, some of them quite large (up to 800 variables), in order to study their behavior in practice.

E-mail: {jgamez,juanlmc,jpuerta}@dsi.uclm.es
Department of Computing Systems. Intelligent Systems and Data Mining Group - $i^3\mathcal{A}$. University of Castilla-La Mancha, 02071 Albacete, Spain.

1 Introduction

During the last two decades there has been an increasing interest in the Bayesian network (BN) formalism (Pearl, 1988; Jensen and Nielsen, 2007; Heckerman, 1997). Among the features that have made BNs a successful formalism are: 1) their mathematical basis is rigorously justified; 2) they deal in an innate way with uncertainty (modeled as a joint probability distribution); 3) they are understandable because of their graphical representation; and 4) they take advantage of locality both in knowledge representation and during inference.

Another advantage of BNs is their capacity for being used both as predictive and descriptive models. In prediction they constitute an efficient tool for solving different inference tasks (posterior probability, abductive or diagnostic reasoning, relevance analysis, classification). As a descriptive tool they possess the ability to efficiently represent the dependence/independence relationships among the random variables that compose the problem domain we wish to model.

Because manual construction of BNs is a complex and highly time-consuming task (the knowledge acquisition bottleneck, probability elicitation, dealing with very large domains, ...), and given the increasing availability of data in many domains, one of the areas that has seen a major activity in BN research is that of automatically learning the BN structure from data. This is the task we focus on in this paper.

Generally speaking, there are two main approaches for structural learning in BNs:

- *Score+search methods.* In these algorithms a function f is used to score a network/DAG (Directed Acyclic Graph) with respect to the training data, and a search method is used to look for the network with the best score. Different Bayesian and non-Bayesian scoring metrics can be used (Neapolitan, 2003; Heckerman et al, 1995; de Campos, 2006). As learning BNs from data is an NP-Hard problem (Chickering, 1996), many heuristics have been proposed to guide the search. In this paper we focus on the application of local search methods to the problem of learning Bayesian networks in the space of DAGs (Buntine, 1991, 1996; Heckerman et al, 1995; Chickering, 2002; de Campos and Puerta, 2001; Friedman et al, 1999; Neapolitan, 2003), although other metaheuristic search methods have also been employed, e.g., genetic algorithms (Larrañaga et al, 1996), simulated annealing (Chickering et al, 1995), tabu search (Acid and de Campos, 2003), ant colony optimization (de Campos et al, 2002), estimation of distribution algorithms (Blanco et al, 2003), etc.
- *Constraint-based methods.* The idea underlying these methods is to satisfy as much independence present in the data as possible (Spirtes et al, 1993; Neapolitan, 2003). Statistical hypothesis testing is used to determine the validity of conditional independence sentences.

There also exist hybrid algorithms that combine these two approaches (Dash and Druzdzel, 1999; Acid and de Campos, 2001) and algorithms that search using an ordering of the variables (Cooper and Herskovits, 1992; WenChen et al, 2008).

As it has been mentioned above, in this paper we focus on local search in the space of DAGs. As the cardinality of this search space is super-exponential (Robinson, 1977), a good idea, specially in domains with a large number of variables, is to limit in some way the areas of the search space to be visited. This idea is not new and has been exploited previously. For example, in (Cano et al, 2004) they restrict the number of possible parents for a variable to be the k most-correlated with it, and the K2 algorithm (Cooper and Herskovits, 1992) is used to learn the BN structure. In a similar way, in (van Dijk et al, 2003) they first construct an undirected graph or skeleton by using zero- and first-order dependence tests, and then a genetic algorithm is employed which is restricted to searching for DAGs belonging to such a skeleton. The approach taken in (Wong and Leung, 2004) is slightly different in that it also uses a first stage based on low order conditional independence tests, but instead of retrieving a skeleton, each possible edge is labeled with the p-value returned by the statistical test; then a genetic algorithm is used during the search phase, and at each generation only those edges with a p-value smaller than a parameter α (also evolved by the genetic algorithm) are considered, but in any case the above algorithms are very difficult to use in large and complex domains. On the other hand, a local method such as Max-Min Hill Climbing (Tsamardinos et al, 2006a) is also a two-step algorithm that in its first stage tries to identify the parents and children of each variable and in the second uses a local search algorithm to look for the network, but with the search restricted to the set of previously-found adjacencies (parents and children). In (Nägele et al, 2007) the first step is carried out as in the previous algorithm but in the second phase substructures are learned using the information gathered in the first phase. In (Friedman et al, 1999) an iterated hill climbing algorithm is proposed that, at each (outer) iteration restricts, the number of candidate parents for each variable to the k most-promising ones, k having the same value for every variable.

As we can observe, the common feature in all the aforementioned algorithms is that all of them use two clearly-separated stages: (1) search space restriction; and (2) running of a search algorithm over the restricted search space. The aim of this paper is to propose a different way of learning BNs, namely by carrying out these two stages simultaneously. Thus, we directly launch a hill climbing algorithm without previously restricting the search space, and then take advantage of the computations carried out at each search step to guess which edges should not be considered from then on. In this way, the search space is pruned progressively as the search advances. Furthermore, our proposal exhibits a nice theoretical property which does not hold in the previously cited algorithms: under the faithfulness condition, our algorithms always return a *minimal I-map*. By learning an I-MAP we get the certainty that all the independence in the learned model, are in fact independence in the distribution encoded by the data we are learning from, so we can be sure that

our posterior conclusions (probabilistic queries) will be right. By learning a *minimal* I-MAP we are sure that there are no spurious relations between the variables in our learned model (e.g. the complete DAG is an I-MAP), and so the inference over it will not be penalized in time because of such extra dependence.

Our experiments confirm that the resulting hill climbing algorithms are faster than the unconstrained/classical version and other state-of-the-art algorithms without excessively degrading the quality of the network discovered.

This paper is structured as follows: We begin in Section 2 with some preliminaries about DAGs and Bayesian networks and in Section 3 we discuss local search in the space of DAGs. Then, in Section 4 we revise some interesting properties of score metrics that are used in our study. Sections 5 and 6 constitute the core of this paper, as they contain a description of the algorithms which constitute the key to our approach. Section 7 is devoted to evaluating these algorithms experimentally. Finally, in Section 8 we present our final conclusions and outline future research.

2 Preliminaries

Bayesian Networks (also known as probabilistic belief networks or causal networks) are graphical models that can efficiently represent and manipulate n-dimensional probability distributions (Pearl, 1988). This representation has two components that respectively codify qualitative and quantitative knowledge:

- A graphical structure, or more precisely a DAG, $\mathcal{G} = (\boldsymbol{V}, \mathbf{E})$, where the nodes in $\boldsymbol{V} = \{X_1, X_2, \ldots, X_n\}$ represent the random variables[1] from the problem we are modeling, and the topology of the graph (the arcs in $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$) encodes conditional (in)dependence relationships among the variables (by means of the presence or absence of direct connections between pairs of variables).
- A set of numerical parameters ($\boldsymbol{\Theta}$), usually conditional probability distributions drawn from the graph structure: For each variable $X_i \in \boldsymbol{V}$ we have a conditional probability distribution $P(X_i|pa(X_i))$, where $pa(X_i)$[2] represents any combination of the values of the variables in $Pa(X_i)$, and $Pa(X_i)$ is the parent set of X_i in $\mathcal{G}$. From these conditional distributions we can recover the joint probability distribution over $\boldsymbol{V}$ thanks to the Markov Condition:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Pa_{\mathcal{G}}(X_i)) \tag{1}$$

[1] We use standard notation, that is, bold font to denote sets and n-dimensional configurations, calligraphic font to denote mathematical structures, upper case for variables or sets of random variables, and lower case to denote states of variables or configurations of states (vectors).

[2] In case of working with different graphs we will use $Pa_{\mathcal{G}}(X_i)$ to clarify the notation.

This decomposition of the joint distribution gives rise to important savings in storage requirements and also enables the performance of probabilistic inference by means of (efficient) local propagation schemes (Jensen and Nielsen, 2007).

Definition 1 A node or variable X is a *collider* in a path π if X has two incoming edges, i. e. the sub path $A \rightarrow X \leftarrow B$ is in π. A collider is also known as a head to head node.

Definition 2 A *v-structure* in a directed graph $\mathcal{G}$ is a subgraph of three nodes (A, X, B) in which X is a collider and A and B are not connected in $\mathcal{G}$.

Definition 3 A path from node X to node Y is *blocked* by a set of nodes $\mathbf{Z}$, if there is a node W on the path for which one of the following two conditions hold:

1. W is not a collider and $W \in \mathbf{Z}$, or
2. W is a collider and neither W nor its descendants are in $\mathbf{Z}$.

A path which is not blocked is *active* or *open*.

Definition 4 Two nodes X and Y are *d-separated* by $\mathbf{Z}$ in graph $\mathcal{G}$ if and only if every path between X to Y is blocked by $\mathbf{Z}$. Two nodes are *d-connected* if they are not d-separated.

We denote by $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}}$ that variables in $\mathbf{X}$ are conditionally independent (through d-separation) of variables in $\mathbf{Y}$ given the set $\mathbf{Z}$ in a DAG $\mathcal{G}$. The same sentence but in a probability distribution p is denoted by $I_p(\mathbf{X}, \mathbf{Y} | \mathbf{Z})$.

Definition 5 A DAG $\mathcal{G}$ is an *I-map* of a probability distribution p if $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}} \Longrightarrow I_p(\mathbf{X}, \mathbf{Y} | \mathbf{Z})$, and is *minimal* if no arc can be removed from $\mathcal{G}$ without violating the I-map condition. $\mathcal{G}$ is a *D-map* of p if $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}} \Longleftarrow I_p(\mathbf{X}, \mathbf{Y} | \mathbf{Z})$.

When a DAG $\mathcal{G}$ is both an I-map and a D-map of p, it is said that $\mathcal{G}$ and p are *isomorphic* models (that is, $\mathcal{G}$ is a *perfect-map* of p) or we say that p and $\mathcal{G}$ are *faithful* to each other (Spirtes et al, 1993; Neapolitan, 2003).

Furthermore, a distribution p is faithful if there exists a graph, $\mathcal{G}$, to which it is faithful. In a faithful BN $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}} \Leftrightarrow I_p(\mathbf{X}, \mathbf{Y} | \mathbf{Z})$.

It is always possible to build a minimal I-map of any given probability distribution p, but some distributions do not admit an isomorphic model (Pearl, 1988).

In general, when learning Bayesian networks from data our goal is to obtain a DAG that is a minimal I-map of the probability distribution encoded by the dataset.

We will assume faithfulness in the rest of the paper. In such cases, we can assume that the terms d-separation and conditional independence are used interchangeably.

3 Learning BNs by local search

The problem of learning the structure of a Bayesian network can be stated as follows: Given a training dataset $D = \{\mathbf{v}^1, \ldots, \mathbf{v}^m\}$ of instances (configurations of values) of $\boldsymbol{V}$, find the DAG $\mathcal{G}^*$ such that

$$\mathcal{G}^* = \arg\max_{\mathcal{G} \in \mathcal{G}^n} f(\mathcal{G} : D) \tag{2}$$

where $f(\mathcal{G} : D)$ is a scoring metric which evaluates the merit of any candidate DAG $\mathcal{G}$ with respect to the dataset D, and $\mathcal{G}^n$ is the set containing all DAGs with n nodes.

Local Search (specifically hill climbing) methods traverse the search space by starting from an initial solution and performing a finite number of steps. At each step the algorithm only considers local changes, i.e. neighbor DAGs, and chooses the one resulting in the greatest improvement in f. The algorithm stops when there is no local change yielding an improvement in f. Because of this greedy behavior the execution stops when the algorithm is trapped at a solution that maximizes f locally the most times rather than globally maximizing it. Different strategies are used to try to escape from local optima: restarts, randomness, etc.

The effectiveness and efficiency of a local search procedure depends on several factors, such as the neighborhood structure considered, the starting solution or the capacity for fast evaluation of candidate subgraphs (neighbors). The neighborhood structure considered is directly related with the operators used to generate neighbors by applying local changes. In BN learning, the usual choices for local changes in the space of DAGs are arc addition, arc deletion and arc reversal. Of course, except in arc deletion we have to take care not to introduce directed cycles in the graph. Thus, there are $O(n^2)$ possible changes, n being the number of variables. With regard to the starting solution, the empty network is usually considered although random starting points or perturbed local optima are also used, specially in the case of an iterated local search.

Efficient evaluation of neighbors/DAGs is based on an important property of scoring metrics: *decomposability* in the presence of full data. In the case of BNs, decomposable metrics evaluate a given DAG as the sum of its node family scores, i.e. the subgraphs formed by a node and its parents in $\mathcal{G}$. Formally, if f is decomposable then:

$$f(\mathcal{G} : D) = \sum_{i=1}^{n} f_D(X_i, Pa_{\mathcal{G}}(X_i)) \tag{3}$$

$$f_D(X_i, Pa_{\mathcal{G}}(X_i)) = f_D(X_i, Pa_{\mathcal{G}}(X_i) : N_{x_i, pa_{\mathcal{G}}(X_i)}) \tag{4}$$

where $N_{x_i, pa_{\mathcal{G}}(X_i)}$ are the statistics of the variables X_i and $Pa_{\mathcal{G}}(X_i)$ in D, i.e. the number of instances in D that match each possible instantiation of X_i and $Pa(X_i)$.

Thus, if a decomposable metric is used, a procedure that changes only one arc at each move can efficiently evaluate the neighbor obtained by this change. This kind of (local) methods can reuse the computations carried out in previous stages, and only the statistics corresponding to the variables whose parents have been modified need to be recomputed. It is clear that a hill climbing algorithm using the operators of arc addition, deletion and reversal, can take advantage of this operation mode, and specifically it will have to measure the following differences when evaluating the improvement obtained by a neighbor DAG:

1. Addition of $X_j \rightarrow X_i$: $f_D(X_i, Pa(X_i) \cup \{X_j\}) - f_D(X_i, Pa(X_i))$
2. Deletion of $X_j \rightarrow X_i$: $f_D(X_i, Pa(X_i) \setminus \{X_j\}) - f_D(X_i, Pa(X_i))$
3. Reversal of $X_j \rightarrow X_i$: It is obtained as the sequence: deletion$(X_j \rightarrow X_i)$ plus addition$(X_i \rightarrow X_j)$, so we compute $[f_D(X_i, Pa(X_i) \setminus \{X_j\}) - f_D(X_i, Pa(X_i))] + [f_D(X_j, Pa(X_j) \cup \{X_i\}) - f_D(X_j, Pa(X_j))]$

Then, at each step the algorithm analyzes all the possible (local) operations, and chooses the one with the highest positive difference[3].

To end this section, it is worth noting that a decomposable score also makes it easier to use a *cache* in which previously computed scores are stored, thus avoiding unnecessary passes over the data. In fact, as is pointed out in (Friedman et al, 1999), it could even be worthwhile to cache not only the score of a given family, but also the counts table, because in this way some new scores can be computed by marginalization instead of performing a new pass over the data. In this study we use the family as index for the cache but we only store the score, not the counts, which results in easier maintenance and the use of far less memory.

Algorithm 1 outlines the hill climbing algorithm for structural learning of Bayesian Networks. Although any DAG ($\mathcal{G}_0$) can be used to initialize the search, usually the empty graph (i.e. a graph with no arcs) is used. In the algorithm, we assume, that each time a family is scored by using $f_D(\cdot)$, it first looks into the cache in order to retrieve the value, and only if this family has not been previously evaluated, does it then actually compute the score by using the dataset (D) and enter it in the cache.

4 Asymptotic behavior of a scoring metric

In section 3 we introduced the concept of a scoring metric to evaluate a DAG $\mathcal{G}$ with respect to a dataset D. In this section we review some (desirable) properties of scoring metrics, most of which are taken from (Chickering, 2002). These concepts will constitute the theoretical basis of our proposal.

Two DAGs are equivalent if the following two conditions hold (Verma and Pearl, 1991):

1. By dropping directions, the resulting undirected graphs are the same.

[3] From now on we will use the term *diff* to refer to that difference of score between the current structure and the structure resulting of applying the considered operation.

Input: D: A dataset defined over variables $\mathbf{V} = \{X_1, \ldots, X_n\}$
Input: $\mathcal{G}_0$: A DAG defined over $\mathbf{V}$ used as the starting point for the search
Output: A DAG $\mathcal{G}$ being the graphical part of network $\mathcal{B}$

$\mathcal{G} \leftarrow \mathcal{G}_0$;
$f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$; `//`$f$`: decomposable scoring metric`
improvement $\leftarrow$ true;
while *improvement* **do**
  improvement $\leftarrow$ false;
  `//neighbors generated by addition`
  For each node X_i and each node $X_j \notin Pa_{\mathcal{G}}(X_i)$ such that $X_j \rightarrow X_i$ does not introduce a directed cycle in $\mathcal{G}$, compute the difference $diff = f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. Store the change which maximizes $diff$ in $\langle change_a, diff_a \rangle$;
  `//neighbors generated by deletion`
  For each node X_i and each node $X_j \in Pa_{\mathcal{G}}(X_i)$, compute the difference $diff = f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. Store the change which maximizes $diff$ in $\langle change_d, diff_d \rangle$;
  `//neighbors generated by reversal`
  For each node X_i and each node $X_j \in Pa_{\mathcal{G}}(X_i)$ such that reversing $X_j \rightarrow X_i$ does not introduce a directed cycle in $\mathcal{G}$, compute the difference $diff = d_1 + d_2$ where d_1 corresponds to $f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ and d_2 corresponds to $f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. Store the change which maximizes $diff$ in $\langle change_r, diff_r \rangle$;
  `//Checking if improvement`
  Let $d^* = max_{k=a,d,r} diff_k$ and $move^*$ its corresponding change;
  if $d^* > 0$ **then**
    improvement $\leftarrow$ true;
    $\mathcal{G} \leftarrow$ apply $move^*$ over $\mathcal{G}$;
    $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}} + d^*$;
  end
end
return $\mathcal{G}$;

Algorithm 1: Structural learning of BNs by using a hill climbing (HC) algorithm

2. The two graphs have the same v-structures.

Definition 6 A scoring metric f is *score equivalent* if for any pair of equivalent DAGs, $\mathcal{G}$ and $\mathcal{G}'$, $f(\mathcal{G} : D) = f(\mathcal{G}' : D)$.

A probability distribution p is contained in a DAG $\mathcal{G}$ if there exists a set of parameter values $\mathbf{\Theta}$ such that the Bayesian network defined by $(\mathcal{G}, \mathbf{\Theta})$ represents p exactly.

Definition 7 (Consistent scoring criterion (Chickering, 2002))
Let D be a dataset containing m iid samples from some distribution p. Let $\mathcal{G}$ and $\mathcal{H}$ be two DAGs. Then, a scoring metric f is *consistent* if in the limit as m grows large, the following two properties hold:

1. If $\mathcal{H}$ contains p and $\mathcal{G}$ does not contain p, then $f(\mathcal{H} : D) > f(\mathcal{G} : D)$
2. if $\mathcal{H}$ and $\mathcal{G}$ contain p, but $\mathcal{G}$ is simpler than $\mathcal{H}$ ($\mathcal{G}$ has less parameters), then $f(\mathcal{G} : D) > f(\mathcal{H} : D)$

This definition means that if two graphs are correct, then the sparser one should receive more merit. This is in the idea of preferring (correct) sparser graphs.

Proposition 1 *(From (Chickering, 2002; Haughton, 1988; Geiger et al, 2001)) The Bayesian Dirichlet equivalent (BDe) metric is score equivalent and consistent.*

Definition 8 (Locally Consistent scoring criterion (Chickering, 2002))
Let D be a dataset containing m iid samples from some distribution p. Let $\mathcal{G}$ be any DAG, and $\mathcal{G}'$ the DAG obtained by adding edge $X_i \rightarrow X_j$ to $\mathcal{G}$. A scoring metric is *locally consistent* if in the limit as m grows large, the following two conditions hold:

1. If $\neg I_p(X_i, X_j | Pa_{\mathcal{G}}(X_j))$, then $f(\mathcal{G} : D) < f(\mathcal{G}' : D)$
2. If $I_p(X_i, X_j | Pa_{\mathcal{G}}(X_j))$, then $f(\mathcal{G} : D) > f(\mathcal{G}' : D)$

This is the main result for our proposal, as we will explain in the next section, because from the concept of local consistency we can (asymptotically) assume that the differences computed by a locally consistent scoring metric f can be used as conditional independence tests over the dataset D. To do this, we have to suppose that D constitutes a sample which is isomorphic[4] to a graph.

Proposition 2 *(Chickering, 2002) The Bayesian Dirichlet equivalent (BDe) metric is locally consistent.*

Apart form BDe, another scoring criterion for which Propositions 1 and 2 hold is the Bayesian Information Criterion (BIC) (Schwarz, 1978).

5 CHC: constrained hill climbing for learning BNs

For the task of structural learning of BNs in the space of DAGs, the hill climbing algorithm with {arc-addition, arc-deletion, arc-reversal} operations is without any doubt the most used algorithm. Its success is due to its ease of implementation, efficiency and the quality of the obtained output, which is a locally optimal solution. To these advantages, we can add another important one under certain assumptions:

Proposition 3 *Let D be a dataset containing m iid samples from some distribution p. Let $\hat{\mathcal{G}}$ be the DAG obtained by running the HC algorithm and taking a DAG $\mathcal{G}_0$ as the initial solution, i.e., $\hat{\mathcal{G}} = HC(\mathcal{G}_0, D)$. If the metric f used to evaluate DAGs in HC is consistent and locally consistent, then $\hat{\mathcal{G}}$ is a* **minimal I-map** *of p in the limit as m grows large.*

[4] In fact, (Chickering, 2002) proves that the isomorphic condition can be relaxed.

Proof First we prove that $\hat{\mathcal{G}}$ is an I-map of p. Let us suppose the contrary, i.e., $\hat{\mathcal{G}}$ is not an I-map of p. Then there is at least one pair of variables X_i and X_j such that $\langle X_i, X_j | Pa_{\hat{G}}(X_i) \rangle_{\hat{\mathcal{G}}}$ and $\neg I_p(X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i))$. Thus, $\hat{\mathcal{G}}$ cannot be a local optimum of f because the addition of arc $X_j \to X_i$ has a positive difference, as follows from the definition of local consistency.
Now we prove the minimal condition. Again let us suppose the contrary, that is, there exists $X_j \in Pa_{\hat{\mathcal{G}}}(X_i)$ such that $I_p(X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i) \setminus \{X_j\})$. If so, $\hat{\mathcal{G}}$ cannot be a local optimum because there is (at least) one deletion operation with a positive difference (local consistency).

Although this result is not particularly surprising, it is interesting to know that under certain conditions the algorithm always returns a model which is a minimal I-map.

Having looked at the advantages of the HC algorithm, we now consider its efficiency, which is the main focus of our study. Thus, although HC has proved to be quite a CPU-time-efficient algorithm for learning BNs when using domains with a small or medium number of variables, things are different when considering domains with a large number of variables. In these cases, the CPU-time requirements increase considerably and it would be of interest to develop an algorithm that maintains the quality-based advantages of HC but which is more efficient. This is our goal in this study and we introduce new HC-based algorithms that work faster but obtain models of similar accuracy to the ones obtained by the original HC algorithm. Our first algorithm is described in the next two subsections.

5.1 CHC: constrained hill climbing

Our idea is to use a hill climbing method for learning Bayesian networks in which we progressively restrict the number of neighbors to be explored, and so evaluated, at each search step. The theoretical basis on which the progressive neighborhood restriction is based, is the concept of the *locally consistent scoring criterion* (def. 8). If we use a locally consistent score metric (e.g. BDe or BIC), we can (asymptotically) use metric differences as a sort of conditional independence test over the dataset D. Thus, we make the following observations:

- Addition. If when adding $X_j \to X_i$ we get $diff < 0$, then (asymptotically) $I_p(X_i, X_j | Pa_G(X_i))$, and so we no longer have to test the addition of X_j as parent of X_i.
- Deletion. This case is analogous to addition. Now, if we get $diff > 0$ when deleting $X_j \to X_i$, then again we have (asymptotically) $I_p(X_i, X_j | Pa_G(X_i))$.

As reversal can be defined as the sequence of deletion+addition, we can study if some comparison can be avoided during the rest of the search in all the basic operations. The way in which we propose to take advantage of this result is by associating a set of *forbidden parents* (FPs) to each node. These sets are initialized to the empty set at the beginning of the search, and then updated

during the search by taking into account the differences computed for each local change tested (addition, deletion and reversal):

- Adding $X_j \to X_i$. If $diff < 0$ then it makes no sense to test this operation anymore, and therefore we add $\{X_j\}$ to $FP(X_i)$.
- Deleting $X_j \to X_i$. If $diff > 0$ then we can remove the arc, and so it makes no sense to test its inclusion anymore. Therefore, we add $\{X_j\}$ to $FP(X_i)$.
- Reversal of $X_j \to X_i$. Decompose as deleting($X_j \to X_i$)+adding($X_i \to X_j$) and use the previous two rules to update (if necessary) the sets $FP(X_i)$ and $FP(X_j)$.

The idea expressed above corresponds to our original design for the CHC algorithm as introduced preliminarily in (Gámez and Puerta, 2005), but as we also showed in the same study the performance of that initial proposal can be improved if edges instead of arcs are forbidden. That is, we proposed the inclusion of X_j in $FP(X_i)$ when, due to the difference obtained in $f_{\mathcal{G}}$, we can (asymptotically) consider conditional independence $I(X_i, X_j|Pa_G(X_i))$. However, because of the symmetry of conditional independence, what this independence says is that X_i is independent of X_j given $Pa_G(X_i)$, so we can discard the undirected link $X_i - X_j$, that is, besides including X_j in $FP(X_i)$ we can also include X_i in $FP(X_j)$. One of the main advantages of this behavior is that the modifications over $FP(\cdot)$ can be exploited in the current iteration, e.g., we can avoid measuring adding $(X_i \to X_j)$ if we obtained a negative difference when measuring adding $(X_j \to X_i)$, as we can observe in Example 1. In this paper, by CHC we always refer to the version in which edges are forbidden. Algorithm 2 shows the description of the CHC algorithm.

Example 1 Let us consider the DAG $\mathcal{G}_t$ shown in Figure 1.(a) as our *true* or target model, and suppose that we get a dataset D by sampling from it. Let us also assume that (as usual) we take the empty graph (Figure 1.(b)) as the starting point for the search. Then, after initializing the forbidden parent sets to empty ($FP(A) = FP(B) = FP(C) = FP(D) = \emptyset$), if we forbid edges as the search progresses then we have to test all the following pairs:

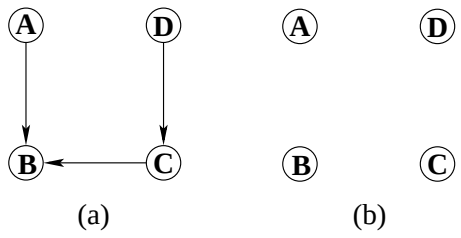

Fig. 1 Graphs for example 1: (a) Target graph. (b) Empty graph.

(1) add $A \to B$: $diff \gg 0$ because $A \to B \in \mathcal{G}_t$, so $\neg I(A, B|\emptyset)_{\mathcal{G}_t}$.
(2) add $A \to C$: $diff < 0$ because $I(A, C|\emptyset)_{\mathcal{G}_t}$: $FP(C) = \{A\}$, $FP(A) = \{C\}$

(3) add $A \rightarrow D$: $\mathit{diff} < 0$ because $I(A, D|\emptyset)_{\mathcal{G}_t}$: $FP(D) = \{A\}$, $FP(A) = \{C, D\}$.
(4) add $B \rightarrow A$: $\mathit{diff} \gg 0$ because $A \rightarrow B \in \mathcal{G}_t$, so $\neg I(A, B|\emptyset)_{\mathcal{G}_t}$.
(5) add $B \rightarrow C$: $\mathit{diff} \gg 0$ because $C \rightarrow B \in \mathcal{G}_t$, so $\neg I(C, B|\emptyset)_{\mathcal{G}_t}$.
(6) add $B \rightarrow D$: $\mathit{diff} > 0$ because $\neg I(B, D|\emptyset)_{\mathcal{G}_t}$.
(-) there is no need to test add $C \rightarrow A$ because $C \in FP(A)$. However, if C had not been added to $FP(A)$ in step (1) then this step could not be skipped.
(7) add $C \rightarrow B$: $\mathit{diff} \gg 0$ because $C \rightarrow B \in \mathcal{G}_t$, so $\neg I(C, B|\emptyset)_{\mathcal{G}_t}$.
(8) add $C \rightarrow D$: $\mathit{diff} \gg 0$ because $D \rightarrow C \in \mathcal{G}_t$, so $\neg I(D, C|\emptyset)_{\mathcal{G}_t}$.
(-) there is no need to test add $D \rightarrow A$ because $D \in FP(A)$. However, if D had not been added to $FP(A)$ in step (3) then this step could not be skipped.
(9) add $D \rightarrow B$: $\mathit{diff} > 0$ because $\neg I(B, D|\emptyset)_{\mathcal{G}_t}$.
(10) add $D \rightarrow C$: $\mathit{diff} \gg 0$ because $D \rightarrow C \in \mathcal{G}_t$, so $\neg I(D, C|\emptyset)_{\mathcal{G}_t}$.

Input: D: A dataset defined over variables $\mathbf{V} = \{X_1, \ldots, X_n\}$
Input: $\mathcal{G}_0$: A DAG defined over $\mathbf{V}$ used as the starting point for the search
Output: A DAG $\mathcal{G}$ being the graphical part of network $\mathcal{B}$

$\mathcal{G} \leftarrow \mathcal{G}_0$;
$f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$; `//f: decomposable scoring metric`
For each X_i do $FP(X_i) = \emptyset$;
improvement $\leftarrow$ true;
while *improvement* **do**
  improvement $\leftarrow$ false;
  `//neighbors generated by addition`
  For each node X_i and each node $X_j \notin (Pa_{\mathcal{G}}(X_i) \cup FP(X_i))$ such that $X_j \rightarrow X_i$ does not introduce a directed cycle in $\mathcal{G}$, compute the difference $\mathit{diff} = f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. If $\mathit{diff} < 0$ then add $\{X_j\}$ to $FP(X_i)$ and add $\{X_i\}$ to $FP(X_j)$. Store the change which maximizes diff in $\langle change_a, \mathit{diff}_a \rangle$;
  `//neighbors generated by deletion`
  For each node X_i and each node $X_j \in Pa_{\mathcal{G}}(X_i)$, compute the difference $\mathit{diff} = f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. If $\mathit{diff} > 0$ then add $\{X_j\}$ to $FP(X_i)$ and add $\{X_i\}$ to $FP(X_j)$. Store the change which maximizes diff in $\langle change_d, \mathit{diff}_d \rangle$;
  `//neighbors generated by reversal`
  For each node X_i and each node $X_j \in Pa_{\mathcal{G}}(X_i)$ such that reversing $X_j \rightarrow X_i$ does not introduce a directed cycle in $\mathcal{G}$, compute the difference $\mathit{diff} = d_1 + d_2$ where d_1 corresponds to $f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ and d_2 corresponds to $f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$. If $d_1 > 0$ or $d_2 < 0$ then add $\{X_j\}$ to $FP(X_i)$ and add $\{X_i\}$ to $FP(X_j)$. Store the change which maximizes diff in $\langle change_r, \mathit{diff}_r \rangle$;
  `//Checking if improvement`
  Let $d^* = max_{k=a,d,r} \mathit{diff}_k$ and $move^*$ its corresponding change;
  if $d^* > 0$ **then**
    improvement $\leftarrow$ true;
    $\mathcal{G} \leftarrow$ apply $move^*$ over $\mathcal{G}$;
    $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}} + d^*$;
  end
end
return $\mathcal{G}$;

Algorithm 2: Structural learning of BNs by using a Constrained Hill Climbing (CHC) algorithm

To end this section there are two remarks which should be made:

- The first one has to do with the use of metric differences as conditional independence tests. In fact, in a more general framework we could use any conditional independence test instead of metric differences to manage the forbidden parent sets. However, as we have already computed the differences during local change evaluation, using the differences instead of conditional independence tests leads to CPU time savings. Furthermore, there exist studies in the literature (Abellán et al, 2006; Moral, 2004; Margaritis, 2003) that show how Bayesian metrics can be used to perform conditional tests and they found that, in general, they are more reliable than traditional chi-square-based conditional tests.
- The second remark is related with the implementation of the forbidden parent list. Thus, although semantically it is more intuitive to start with no forbidden parents and to add those discovered during the search process, from the implementation point of view it is better to consider a list of *allowed parents*, as at each step the algorithm just runs over the content of this list, instead of having to check whether a possible change involves a forbidden parent or not. So we initialize the lists with all the possible variables, and delete from them those parents which are introduced in FPs.

5.2 Does CHC return a minimal I-map?

CHC retains some of the theoretical properties exhibited by HC, thus, as at each step of CHC we choose the best operation with respect to improving f, *monotonicity* is ensured, i.e., $f(\mathcal{G} : D) \leq f(\mathcal{G}' : D)$, where $\mathcal{G}'$ is the neighbor of $\mathcal{G}$ which maximizes the difference *diff*. Furthermore, because of this monotonic behavior and the fact that CHC stops when there is no neighbor of $\mathcal{G}$ which improves $f(\mathcal{G})$, *termination* is guaranteed.

However, there are (as expected) some differences between CHC and HC with respect to the quality of the output obtained, even when starting from the same initial point. This difference in behavior is due to the action of constraining the set of forbidden parents for each variable during the search in CHC. As we have mentioned above, CHC relies on the conditions required in the definition of *local consistency* (def. 8), which ensure asymptotic behavior but rarely hold in real datasets. Because of this, CHC can get stuck in a locally sub-optimal solution while HC gets stuck in a locally optimal solution. Furthermore, in CHC we cannot ensure that $\hat{\mathcal{G}} = CHC(\mathcal{G}_0)$ is, asymptotically, an I-map of p. To support these claims let us consider the following two examples:

Example 2 Let us consider Figure 2 in which we show the evolution of the search process for two different datasets[5] (alarm and pigs) with 37 and 441 variables respectively. We have selected these two networks because they represent the extremes in terms of size with respect to the set of networks used in

[5] See Section 7 for a complete description of these networks/datasets

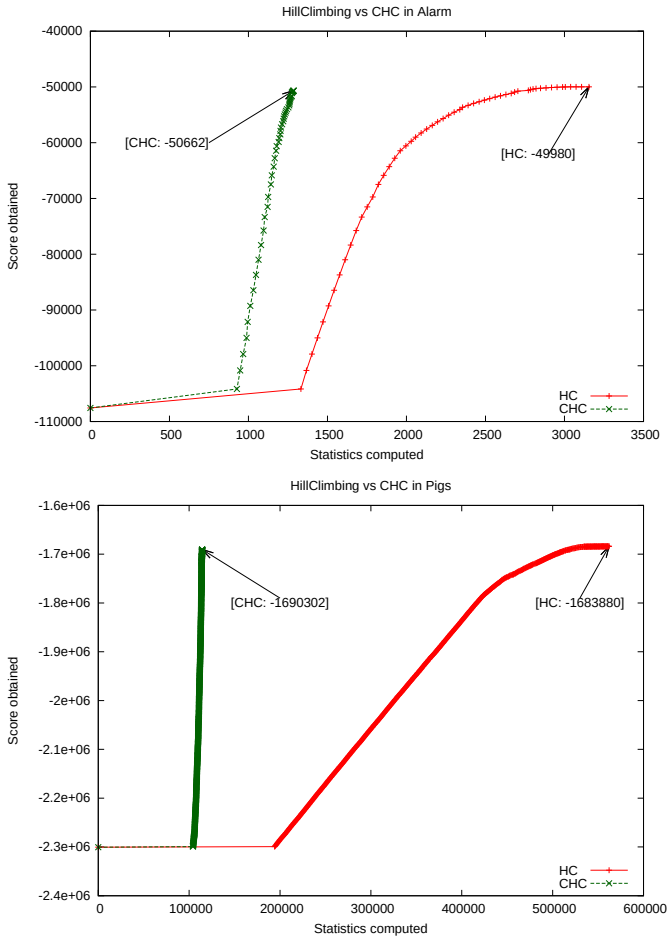

Fig. 2 Evolution of the search process in *alarm* (above) and *pigs* (below) datasets when using HC and CHC algorithms. The graphs represent the number of computed statistics (x-axis) vs quality of the obtained model (y-axis). Each point corresponds to a complete search step, i.e. analysis of all the possible local changes for the current graph.

this study. The behavior with other networks can be assumed to be similar. In both cases the search starts from the empty network and as we can see in both cases CHC needs to compute fewer statistics than HC but obtains a DAG of worse quality. Note also how CHC takes advantage of FP sets from the first iteration, avoiding studying the $O(n^2)$ local changes (additions), which is what HC has to do.

Example 3 Let us retake Example 1. After the first step we have:

$$FP(A) = \{C, D\} \quad FP(C) = \{A\}, \quad FP(D) = \{A\},$$

and because of the presence of a direct dependence in the original (target) model, we can assume that the highest positive difference corresponds to one

of the following arcs: $\{A \to B, B \to A, B \to C, C \to B, C \to D, D \to C\}$. Let us suppose that as a result of the sampling process the highest positive difference corresponds to arc $B \to C$, that is, good edge but bad direction. Then, the new graph is shown in Figure 3.(b).

In the second step of CHC most of the statistics are taken from the cache, but we must test those involving C because its family has changed. Thus, we can observe how the reversal of $B \to C$ makes no sense because it was the last arc added. Addition of $A \to C$ is not tested because $A \in FP(C)$. Finally, we have to test $D \to C$. This arc will receive a positive difference (because in fact there is a direct relation between D and C in the true model), but this difference will be smaller than in the previous step, because now it introduces a marginal independence between D and B that does not hold in the true model. Therefore, after analyzing these local changes, the candidate arcs to be added must be $\{C \to D, A \to B, B \to A\}$. Let us suppose that the algorithm chooses the first one, thus giving the updated model shown in Figure 3.(c).

In the third step neither reversal nor deletion of $B \to C$ or $C \to D$ makes sense. We cannot test the addition of $A \to D$ because $A \in FP(D)$, or of $D \to B$ because it introduces a cycle. So, the only new addition to be considered is $B \to D$, but this change will receive a negative difference because it breaks the conditional independence sentence of B and D given C, which is present in the target model. Therefore we update $FP(D) = \{A, B\}$ and $FP(B) = \{D\}$. Thus, the only two arcs with a positive difference are $\{A \to B, B \to A\}$. The algorithm can freely choose either of them because both introduce the same conditional independence sentence $(I(A, C|B))$, so let us suppose that the second one is chosen (see Figure 3.(d)).

At this point, neither deletion nor reversal will obtain a better model. Furthermore, no addition can be studied because of the content of FP sets, so the algorithm ends up returning the current DAG (Figure 3.(d)), which is not an I-map of the one in Figure 3.(a) because it contains the conditional independence between A and C given B, which does not appear in the original graph.

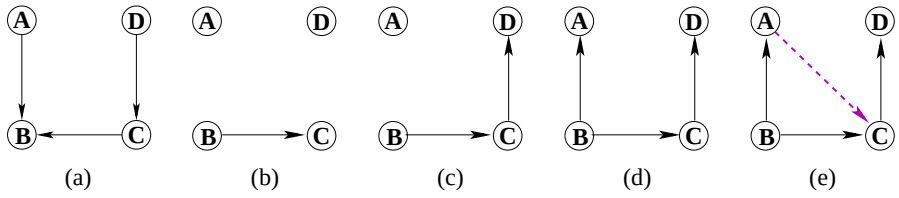

Fig. 3 Graphs for Example 3: (a) Target graph. (b) DAG after first step. (c) DAG after second step. (d) DAG after third step (result of CHC). (e) DAG representing a minimal I-map of (a).

With the two previous examples we have tried to show how CHC can return a non minimal I-map and/or a sub-optimal solution. Notice that following

Example 3, it is easy to see how an unconstrained hill climbing algorithm will test the addition of arc $A \rightarrow C$, obtaining in this way a minimal I-map of the initial model (Figure 3.(e)). To solve these problems we propose the use of the output of CHC as the input for unconstrained hill climbing (Algorithm 1).

The resulting algorithm, denoted by CHC* (Algorithm 3), (obviously) has the same properties as HC and will (probably) improve the quality of the solution obtained by CHC at (hopefully) a small cost, because as we start the unconstrained process from a very good starting point, few iterations are likely to be needed to get a locally optimal solution. Figure 4 shows a graphical illustration of the expected behavior over the two networks considered above. As we can see, in both cases the quality of the obtained networks improves with the subsequent application of the unconstrained search, even surpassing HC in the case of the larger network. On the other hand, the number of statistics computed also increases, but it is still lower (significantly in the larger network) than when applying HC from the beginning.

Input: D: A dataset defined over variables $\mathbf{V} = \{X_1, \ldots, X_n\}$
Input: $\mathcal{G}_0$: A DAG defined over $\mathbf{V}$ used as the starting point for the search
Output: A DAG $\mathcal{G}$ being the graphical part of network $\mathcal{B}$

$\mathcal{G}_1 \leftarrow CHC(D, \mathcal{G}_0)$;
$\mathcal{G} \leftarrow HC(D, \mathcal{G}_1)$;
return $\mathcal{G}$;

Algorithm 3: Structural learning of BNs by using a *constrained* hill climbing followed by an unconstrained hill climbing algorithm (CHC*)

Proposition 4 *Under the same conditions as Proposition 3, CHC* (Algorithm 3) returns a* **minimal I-map**.

Proof Follows directly from Proposition 3 because there is no restriction on the graph considered as the starting point for HC.

6 Iterated CHC

In the previous section we introduced the CHC algorithm, showed its problems and solved them by adding a second phase which consists of an unconstrained search. Although the resulting algorithm meets all the requirements we set at the beginning of the paper, namely, it is faster than directly using HC and it retains the nice theoretical properties of the DAG returned, it has the *conceptual* disadvantage of having to use an unconstrained step in a *constrained* proposal. Furthermore, the computational load, which was even smaller than using only HC, significantly increases (with respect to CHC) with the unconstrained step. In this section, we propose two algorithms that use CHC as a

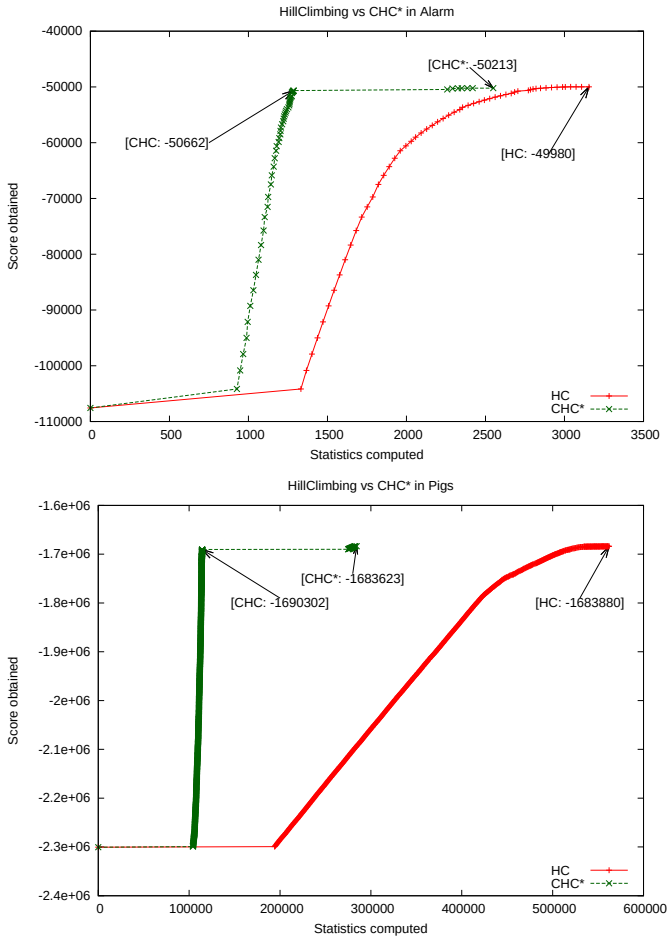

Fig. 4 Evolution of the search process in *alarm* (above) and *pigs* (below) datasets when using HC and CHC* algorithms. The graphics represent the number of computed statistics (x-axis) vs quality of the obtained model (y-axis). Each point corresponds to a complete search step, i.e. analysis of all the possible local changes for the current graph.

building block and also satisfy the aforementioned requirements but without having to use the unconstrained phase.

The first one consists of the iteration of CHC by taking as input the output obtained in the previous iteration. Because FP sets are reset at the beginning of each iteration, this fact allows the algorithm to escape from the current point as an unconstrained HC will do. However, running CHC again takes advantage of FP sets and so the number of computed statistics is smaller than that required by launching an unconstrained HC. On the other hand, the algorithm stops when no modification is carried out over the DAG returned by the previous iteration. When this happens, and because FP sets are empty at the beginning, we can be sure that HC will also stop without making any modifi-

cation, and so we get a minimal I-map. Algorithm 4 (iCHC) and Proposition 5 show the details.

Input: D: A dataset defined over variables $\mathbf{V} = \{X_1, \ldots, X_n\}$
Input: $\mathcal{G}_0$: A DAG defined over $\mathbf{V}$ used as the starting point for the search
Output: A DAG $\mathcal{G}$ being the graphical part of network $\mathcal{B}$

$\mathcal{G} \leftarrow \mathcal{G}_0$;
$f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$; `//f: decomposable scoring metric`
improvement $\leftarrow$ true;
while *improvement* **do**
  improvement $\leftarrow$ false;
  $\mathcal{G}_1 = CHC(D, \mathcal{G})$;
  $f_{\mathcal{G}_1} \leftarrow f(\mathcal{G}_1 : D)$;
  if $(f_{\mathcal{G}_1} > f_{\mathcal{G}})$ **then**
    improvement $\leftarrow$ true;
    $\mathcal{G} \leftarrow \mathcal{G}_1$;
    $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}_1}$;
  end
end
return $\mathcal{G}$;

Algorithm 4: Structural learning of BNs by using an iterated *constrained* hill climbing algorithm (iCHC)

Proposition 5 *Under the same conditions as Proposition 3, iCHC (Algorithm 4) returns a* **minimal I-map**.

Proof Notice that iCHC finishes when no change is applied by CHC to the incoming DAG. As the initial step is equivalent to the one carried out by HC (as CHC starts with empty FP sets), we can conclude that iCHC will also stop without carrying out any change over the incoming DAG. Therefore, applying Proposition 3, the output is a minimal I-map.

In preliminary experiments not included in the paper, we observed that iCHC finds the best model at the second iteration and so finishes at the third one after checking that no change has been made. Of course, this is not the general case, and sometimes the algorithm carries out more iterations. However, as we know that the algorithm can (sometimes) improve the score of the DAG by performing more iterations, we can ask ourselves the following question: *is it enough with two iterations of CHC to guarantee the minimal I-map condition on the returned graph?* If the answer is *yes*, then a two-iteration CHC algorithm or 2iCHC will be of interest in itself, because (1) it meets all our requirements for the quality of the returned model, and (2) it is faster than iCHC, because we can stop after just the second iteration without the need to test in the third iteration that no change has been made (that, of course, needs some extra computations). 2iCHC is shown in Algorithm 5 and the positive answer to our previous question is justified in Proposition 6.

Input: D: A dataset defined over variables $\mathbf{V} = \{X_1, \ldots, X_n\}$
Input: $\mathcal{G}_0$: A DAG defined over $\mathbf{V}$ used as the starting point for the search
Output: A DAG $\mathcal{G}$ being the graphical part of network $\mathcal{B}$

$\mathcal{G}_1 \leftarrow CHC(D, \mathcal{G}_0)$;
$\mathcal{G} \leftarrow CHC(D, \mathcal{G}_1)$;
return $\mathcal{G}$;

Algorithm 5: Structural learning of BNs by using a two-iteration *constrained* hill climbing algorithm (2iCHC)

Proposition 6 *Under the same conditions as Proposition 3, 2iCHC (Algorithm 5) returns a* **minimal I-map**.

In this case, we shall leave the proof to the appendix. The idea behind it is supported by the fact that the second iteration starts with all $FP()$ sets empty and all the "true" adjacencies are known in the current graph. The algorithm only has to find the erroneous v-structures in the result and to make the arcs in the v-structure covered by making the extreme nodes adjacent. For example, in Figure 3(d), in the second iteration of the algorithm, $FP(A)$ is empty and $FP(C)$ is also empty, so the algorithm is able to put the arc between the nodes A and C in order to make the collider B; $A \rightarrow B \leftarrow C$ covered (Figure 3(a)).

In Figure 5 the behavior of these two proposed algorithms is shown against CHC* for the two networks considered above. It can be observed that, in both cases, after the second iteration no improvement is made. The computational saving can be easily seen in both cases, but specially in the larger network, while the results obtained are the same in *pigs* and only slightly lower for *alarm*.

7 Experimental Evaluation

In this section we describe a set of experiments aimed at testing the performance of the algorithms presented in this paper. Given the nature of the proposed algorithms, we were not only interested in the quality of the solution obtained by them but also in their computational complexity. Below we provide details about the implementation of the algorithms, the datasets used in this comparison, the indicators we chose to argue about the goodness of each algorithm, and to end the section we give the results and their analysis.

7.1 Algorithms

The algorithms examined in this section are the standard hill climbing (HC), the constrained hill climbing followed by unconstrained hill climbing (CHC*), iterated constrained hill climbing (iCHC), and two-iteration constrained hill climbing (2iCHC) algorithms. In all cases an empty network is used as the starting graph. We do not consider the constrained hill climbing algorithm

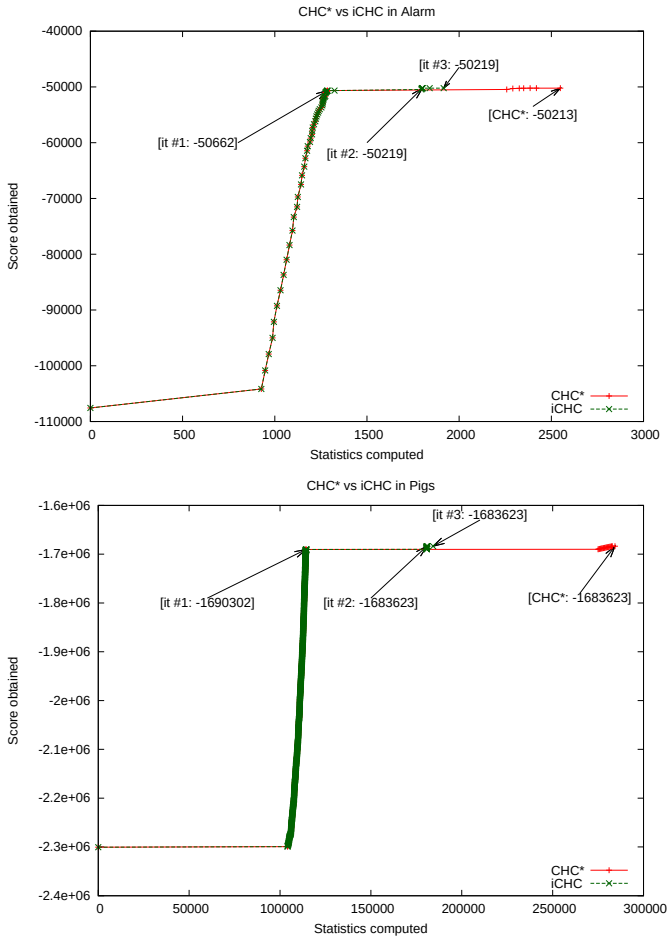

Fig. 5 Evolution of the search process in *alarm* (above) and *pigs* (below) datasets when using CHC* and the iterated CHC algorithms (iCHC and 2iCHC). The graphics represent the number of computed statistics (x-axis) vs quality of the obtained model (y-axis). Each point corresponds to a complete search step, i.e. analysis of all the possible local changes for the current graph.

(CHC) because it does not guarantee a minimal I-map of the original distribution. In addition we consider the Max-Min Hill Climbing algorithm (MMHC) (Tsamardinos et al, 2006a), as it being the best scalable state-of-the-art structural learning algorithm to deal with a high number of variables.

An implementation of MMHC is available in Matlab from the authors' web page. However, access to the source code is not free. As the functionality provided by that implementation is limited and also the second stage of the algorithm is based on a tabu search rather than canonical hill climbing, a custom implementation is used here. This implementation is based on the implementation of the MMPC algorithm available from (Peña et al, 2006) with

the efficiency improvements described in (Tsamardinos et al, 2006a). For the second stage of the algorithm, the implementation of hill climbing, also used in this comparison, is employed.

The score metric used here is the Bayesian Dirichlet equivalent uniform (BDeu) (Heckerman et al, 1995) with the same parameters as in (Tsamardinos et al, 2006a), i.e. equivalent sample size equal to 10.

The actual implementation was coded in Java and interacts with the WEKA library (Witten and Frank, 2005) for dataset management.

In order to speed up all the algorithms we use an internal cache where we save the result of every score computation in order to re-use it later in the execution. This cache is implemented with a hash table, so updating and querying operations over it are linear in time. All the runs of these algorithms were carried out on a dedicated server with Pentium Xeon 3.0 Ghz, 64 bits architecture, 4 Gb RAM memory and under Linux. The Java Runtime Environment used is the one provided by SUN Microsystems, version 1.5.

7.2 Networks

For this experimental comparison we selected the 22 networks used in (Tsamardinos et al, 2006a). These networks comprise 8 well-known models for different real-world applications, ALARM (Beinlich et al, 1989), BARLEY (Kristensen and Rasmussen, 2002), CHILD (Cowell et al, 2003), HAILFINDER (Jensen and Jensen, 1996), INSURANCE (Binder et al, 1997), MILDEW (Jensen and Jensen, 1996), MUNIN version 1 (Andreassen et al, 1989) and PIGS (Jensen, 1997). For some of these networks (ALARM, CHILD, HAILFINDER and INSURANCE) an augmented version has been built by joining together 3, 5 or 10 copies of the original network as described in (Tsamardinos et al, 2006b; Statnikov et al, 2003). Finally, we also used a network learned using the Sparse Candidate (SC) algorithm (Friedman et al, 2000) on a dataset about gene expression micro-arrays (Spellman et al, 1998), and this network will be called GENE. In all cases, both network files and sampled datasets were obtained from the Discovery Systems Laboratory web site[6].

We can see more detailed information about these networks in Table 1.

For each of these networks we work with datasets sampled from them with three different sizes, 500, 1000 and 5000 instances, therefore 66 ($3\times$ 22) training sets have been used. For validation, 5 independent test sets were sampled for each case, that is, five test sets for (Alarm1,500), five for (Alarm1,1000), etc. The reported results are the average obtained over the five test sets.

7.3 Performance indicators

As performance indicators we consider two kinds of factors, one being the quality of the network obtained by the algorithm, and the other the complexity

[6] http://www.dsl-lab.org/

Table 1 Bayesian networks used in our experimental evaluation. For each network we indicate the number of variables, edges, minimum/maximum number of states, average/maximum number of parents per variable and average/maximum number of parents and children (PC) per variable.

	vars	edges	min-max states	average parents	max. parents	average PC	max. PC
Alarm1	37	46	2-4	1.24	4	2.49	6
Alarm3	111	149	2-4	1.34	4	2.68	6
Alarm5	185	265	2-4	1.43	4	2.86	8
Alarm10	370	570	2-4	1.54	4	3.08	9
Barley	48	84	2-67	1.75	4	3.50	8
Child	20	25	2-6	1.25	2	2.50	8
Child3	60	79	2-6	1.32	3	2.63	8
Child5	100	126	2-6	1.26	2	2.52	8
Child10	200	257	2-6	1.29	2	2.57	8
Gene	801	972	3-5	1.21	4	2.43	11
HailFinder	56	66	2-11	1.18	4	2.36	17
HailFinder3	168	283	2-11	1.68	5	3.37	19
HailFinder5	280	458	2-11	1.64	5	3.27	19
HailFinder10	560	1017	2-11	1.82	5	3.63	21
Insurance	27	52	2-5	1.93	3	3.85	9
Insurance3	81	163	2-5	2.01	4	4.02	9
Insurance5	135	284	2-5	2.10	5	4.21	10
Insurance10	270	556	2-5	2.06	5	4.12	11
Link	724	1125	2-4	1.55	3	3.11	17
Mildew	35	46	3-100	1.31	3	2.63	5
Munin1	189	282	1-21	1.49	3	2.98	15
Pigs	441	592	3-3	1.34	2	2.68	41

of each algorithm. In the first group we take into account the value of the score metric (BDeu) for the resulting model and also the average likelihood of that model given five different test datasets, each one with the same number of instances as the corresponding training dataset. In addition we take into account the Structural Hamming Distance (SHD) with respect to the golden model (the one used to sample training and test sets), also measured as in (Tsamardinos et al, 2006a) and based on comparing the essential graphs.

The Structural Hamming distance is computed for a pair of partially directed acyclic graphs (PDAG). Thus, the output models obtained by each algorithm are converted to a PDAG. Then, for every pair of graphs, SHD is computed as the number of edges missing in the first graph wrt. the second, plus the number of extra edges in the first graph wrt. the second, and plus the number of edges present in both graphs but with different directions. This last set of edges includes the reversed edges and also the edges that are undirected in one of them and directed in the other.

Regarding the second factor, we collect the number of computations carried out by each algorithm, i.e. statistical tests or score function computed, which have a direct correspondence with CPU time required. We named this factor as *calls*. In this case we do not use the runtime used by every algorithm, mainly because the MMPC algorithm is implemented in C++ and the rest of the

Table 2 Averaged BDeu score relative to the 2iCHC algorithm. The best results are highlighted in bold type. The • symbol indicates those results not significantly worse than the best one.

	Sample size			
	500	1000	5000	averaged
HC	**0.988**	**0.989**	0.991	**0.989**
CHC*	•0.990	•0.992	**0.986**	**0.989**
iCHC	0.998	0.998	0.996	0.997
2iCHC	1.000	1.000	1.000	1.000
MMHC	1.010	1.010	1.061	1.027

algorithms are implemented in Java. Therefore, knowing that running C++ code is more efficient than running Java code, the runtime results cannot be used properly because they are biased in favor of the MMHC algorithm.

7.4 Empirical results

In this section we present, in the following tables, a summary of the results we obtained after running the five algorithms over the 22 datasets for better readability. The complete set of results are available in appendix B. In all these tables we show the averaged results over all datasets and relative to the figures for the 2iCHC algorithm. The MMHC algorithm could not handle MUNIN1 with sample sizes 1000 and 5000 and MILDEW with 5000 instances with the available memory, so results for such sizes are averaged over 21 or 20 datasets instead of 22. For each one of the indicators we performed a Friedman test (Friedman, 1937) in order to discover whether we can assume that all algorithms show the same performance and in the case that we cannot assume so we use Holm's post-hoc test (Holm, 1979) to see which algorithms can be taken as not statistically worse than the best one. In all cases we use the standard significance level $\alpha = 0.05$.

First, in Table 2 we can see the results for the BDeu score of the learned model. Results show that the best algorithms are hill climbing and CHC*, MMHC being the worst. According to the Friedman test, all algorithms are not comparable for any of the sample sizes and Holm's test says that only CHC* can be considered as good as HC with 500 and 1000 instances, but no other algorithm can be compared to CHC* in the dataset with 5000 instances.

Table 3 shows the results for the likelihood. For each learned model, we compute the log-likelihood over the five sampled test sets, and average the results. From here we can draw a similar conclusion as before, since again the best algorithms are HC and CHC* and again the worst one is MMHC. The Friedman test says that not all algorithms are equally good and the post-hoc test concludes that only HC and CHC* are not different statistically regardless of sample size. Furthermore, iCHC is not different from the others in the biggest datasets.

In Table 4 we have the results for SHD, from which we can see that according to this indicator the best algorithm is MMHC. Statistical tests say

Table 3 Log-Likelihood of the five test sets for each one of the learned models averaged for all the datasets. The figures are relative to the 2iCHC algorithm. The best results are highlighted in bold type. The • symbol indicates those results not significantly worse than the best one. Originally, log-likelihood values are negative and the higher the better. However, since we report here relative values of negative numbers, in this table a lower value should be considered as better.

	Sample size			
	500	1000	5000	averaged
HC	**0.977**	**0.984**	•0.988	**0.983**
CHC*	•0.980	•0.985	**0.983**	**0.983**
iCHC	0.996	0.996	•0.994	0.996
2iCHC	1.000	1.000	1.000	1.000
MMHC	1.009	1.014	1.071	1.032

that no other algorithm can be considered comparable with MMHC except CHC* with the biggest sample size. This result could indicate that MMHC is better at recovering *causal* relations, probably due to the preprocessing in which it tries to identify the parents and children of each node. However, we must take into account here that the HC and CHC algorithms are not designed to learn causal-like models, but to recover an I-map of the probability distribution encoded in the data. In any case, we have observed that the networks obtained by the MMHC algorithm are sparser than the gold standard and the ones obtained from the rest of the algorithms. MMPC's behavior could bias the SHD measure in its favor. To explain this last sentence we should observe figure 6. Considering the first network as reference, the empty model obtains better merit according to SHD than the model (c), which is a minimal I-map of the reference model.

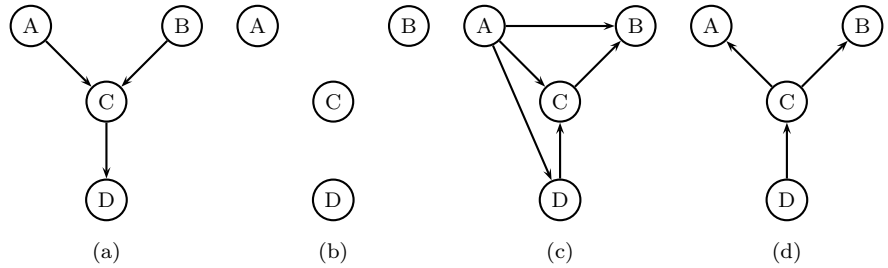

Fig. 6 Example of the bias of SHD. (a) represents the reference model. (b) is the empty network which has a SHD value of 3. (c) is a minimal I-map of (a) but its SHD value is 4. (d) is a network with all arcs reversed with respect to (a) and its SHD is 3.

In general, empty or sparse graphs will receive higher merit according to SHD than a model in which the I-map condition is achieved by covering wrongly discovered v-structures, as the hill-climbing family of algorithms does. If this is the case, the sparser model will not be able to represent the original joint probability distribution, while the I-map model will.

Table 4 Averaged Structural Hamming Distance relative to the 2iCHC algorithm. The best results are highlighted in bold type. The • symbol indicates those results not significantly worse than the best one.

	Sample size			
	500	1000	5000	averaged
HC	0.952	0.941	1.023	0.972
CHC*	0.956	0.962	•0.946	0.955
iCHC	0.996	0.999	0.982	0.992
2iCHC	1.000	1.000	1.000	1.000
MMHC	**0.878**	**0.795**	**0.8045**	**0.826**

Let us now consider the MMHC algorithm, knowing that it restricts the candidate parents for the second phase of the algorithm to only adjacent variables discovered in the first stage. Then, even assuming the MMPC algorithm can correctly discover the set of variables marginally dependent on a given one, there is no way to properly assess the direction of a link between two variables at the beginning of the search because both arcs would have the same merit if the metric used is score equivalent. Then, a possible output of the MMHC algorithm to the model in 6(a) could be the structure in 6(d). In this case the output is not an I-map of the original model and it is totally wrong from a causal point of view, however yet the SHD indicates that this last structure is better than the I-map network.

In the experimentation MMHC has yielded a reasonably good performance in terms of SHD, but its results cannot be guaranteed in any way, there is no theoretical proof of its performance. This means that in some cases the result provided by MMHC can be the opposite of the one expected. On the other hand, the algorithms presented in this paper do have a theoretical proof, thus assuring a minimum of quality in the final result.

Finally, in Table 5 we have results for efficiency. We can see that the most efficient algorithm is 2iCHC and on average the worst one is MMHC. In this case a statistical test does not make much sense as the number of calls for 2iCHC is always smaller than the number of calls for the other algorithms, only iCHC is equal for a few datasets. However, we have performed the statistical analysis and reported the results in the same way as above with the aim of presenting all the result tables in a coherent way. It is interesting to notice that all algorithms tend to be more inefficient compared to 2iCHC as the amount of data increases. However, in the case of MMHC this trend is notably sharper. The reason why the MMHC algorithm seems to be more inefficient with the larger dataset is due to the result of the Pigs dataset, because as only in the first phase of this algorithm the number of calls is 20 times greater than the total calls needed for the HC algorithm. This result is justified because as the amount of data increases the statistical tests are more reliable and in the worst case scenario the complexity of the algorithm is exponential. Notice that the set of parents and children for one variable in the Pigs dataset is really large, namely 41.

Table 5 Averaged number of calls relative to the 2iCHC algorithm. The best results are highlighted in bold type. The • symbol indicates those results not significantly worse than the best one.

	Sample size			
	500	1000	5000	averaged
HC	2.137	2.273	2.425	2.278
CHC*	1.610	1.631	1.656	1.633
iCHC	•1.055	•1.068	1.079	1.067
2iCHC	**1.000**	**1.000**	**1.000**	**1.000**
MMHC	1.137	1.283	7.049	3.156

Table 6 Averaged ratio between the gain in likelihood obtained by each model with respect to the empty model and the number of computations used. The value is also relative to the 2iCHC algorithm. The best results are highlighted in bold type. The • symbol indicates those results not significantly worse than the best one.

	Sample size			
	500	1000	5000	averaged
HC	0.548	0.534	0.505	0.529
CHC*	0.693	0.674	0.680	0.682
iCHC	•0.978	•0.971	0.942	0.964
2iCHC	**1.000**	**1.000**	**1.000**	**1.000**
MMHC	0.628	0.554	0.400	0.527

If we take a look at these results globally we can conclude that in terms of the accuracy of the learned model the family of CHC algorithms is between the performance of HC and MMHC, but with relatively small differences. However, in terms of efficiency, the improvement of these algorithms, especially 2iCHC, is considerable greater than HC and MMHC. Additionally, under the considered assumptions, all the proposed algorithms return a minimal I-map, while MMHC does not.

To perform a better analysis of the trade-off between quality and efficiency, the ratio between the difference in likelihood for each learned model and the empty network and the number of computations each algorithm needs to obtain such a gain has been analyzed. It can be expressed as $\frac{LL(M_{i,j})-LL(Empty_j)}{\#computations_{i,j}}$, where $M_{i,j}$ is the model obtained by the algorithm i on the dataset j, $Empty_j$ is the empty model estimated for the dataset j, and $\#computations_{i,j}$ is the number of calls made by the algorithm i on the dataset j.

This factor says how much benefit is yielded by each algorithm per computation made. Table 6 shows this measure relative to 2iCHC. It lets us argue whether the (small) differences in accuracy are justified by the improvement in efficiency, and although it can be guessed by looking through the two indicators used to compute it, it is easier to reach a conclusion using only this compound factor.

Now, a bigger value means better performance, so it is obvious that the best algorithm is 2iCHC. According to the Friedman test the hypothesis that all algorithms have similar results has to be rejected, as expected. The Holm

test indicates that only iCHC is comparable with 2iCHC, but only in the smallest sample sizes, not with 5000 instances.

8 Conclusions

In this paper we have proposed improved versions of the standard hill climbing algorithm for learning Bayesian networks. This new family of algorithms carries out a constrained hill-climbing search, which consists of the use of a standard hill climbing algorithm as the search engine but restricting the neighborhood by using a list of forbidden parents for each variable. The main novelty here is the use of these forbidden parent lists and that at each iteration they are updated in order to theoretically ensure that the output of the proposed algorithm is a minimal I-map. The underlying idea of the method relies on the theoretical property of local consistency exhibited by some scoring metrics, such as BDe, MDL and BIC.

The validation of our approach is twofold. First we have provided theoretical results for the quality of the output obtained by the algorithms presented, that is, a minimal I-map. Second, we have designed a set of computational experiments over different domains (some of them quite large) in order to test the validity of our approach when dealing with real data. From these experiments we can conclude that our algorithms are (considerably) faster than the standard hill climbing algorithm and Max-Min HC, but with comparable quality of the models returned. With regard to efficiency, we should point out that the gain is specially noteworthy in the two CHC iterated versions. Therefore, our main conclusion is that when the domain complexity increases, iCHC and 2iCHC can be of great utility because they can obtain an accurate Bayesian network model while using little CPU time.

For future work, we are considering the possibility of further reducing the complexity of the second iteration in the CHC family of algorithms. We have realized that the first step of the second iteration is by far the most complex step of the algorithm, even though it is the initial step. This is due to the computation of the new statistics with many more variables involved, as the second iteration does not start from an empty network. So, we will investigate this more deeply in order to avoid the second iteration of the CHC algorithm while maintaining the good properties, empirical and theoretical, of the algorithms presented in this paper.

Acknowledgments

We are grateful to the anonymous reviewers for their valuable comments and suggestions that have greatly helped us to improve the paper. This work has been partially supported by the Spanish Ministerio de Educación y Ciencia (TIN2007-67418-C03-01); Junta de Comunidades de Castilla-La Mancha (PBI-08-048) and FEDER funds.

References

Abellán J, Gómez-Olmedo M, Moral S (2006) Some variations on the PC algorithm. In: Proceedings of the 3rd European Workshop on Probabilistic Graphical Models (PGM-06), pp 1–8

Acid S, de Campos LM (2001) A hybrid methodology for learning belief networks: Benedict. International Journal of Approximate Reasoning 27(3):235–262

Acid S, de Campos LM (2003) Searching for bayesian network structures in the space of restricted acyclic partially directed graphs. Journal of Artificial Intelligence Research 18:445–490

Andreassen S, Jensen FV, Andersen SK, Falck B, Kjærulff U, Woldbye M, Sørensen AR, Rosenfalck A, Jensen F (1989) MUNIN — an expert EMG assistant. In: Desmedt JE (ed) Computer-Aided Electromyography and Expert Systems, Elsevier Science Publishers, Amsterdam, chap 21

Beinlich IA, Suermondt HJ, Chavez RM, Cooper GF (1989) The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. In: Second European Conference on Artificial Intelligence in Medicine, Springer-Verlag, Berlin, vol 38, pp 247–256

Binder J, Koller D, Russell SJ, Kanazawa K (1997) Adaptive probabilistic networks with hidden variables. Machine Learning 29(2-3):213–244

Blanco R, Inza I, Larrañaga P (2003) Learning bayesian networks in the space of structures by estimation of distribution algorithms. International Journal of Intelligent Systems 18(2):205–220

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

Buntine WL (1991) Theory refinement on bayesian networks. In: Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence, pp 52–60

de Campos LM (2006) A scoring function for learning bayesian networks based on mutual information and conditional independence tests. Journal of Machine Learning Research 7:2149–2187

de Campos LM, Puerta JM (2001) Stochastic local algorithms for learning belief networks: Searching in the space of the orderings. In: 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01), pp 228–239

de Campos LM, Fernández-Luna JM, Gámez JA, Puerta JM (2002) Ant colony optimization for learning bayesian networks. International Journal of Approximate Reasoning 31(3):291–311

Cano R, Sordo C, Gutiérrez JM (2004) Applications of bayesian networks in meteorology. In: Gámez JA, Moral S, Salmerón A (eds) Advances in Bayesian Networks, Springer-Verlag, pp 309–327

Chickering DM (1996) Learning Bayesian networks is NP-Complete. In: Fisher D, Lenz H (eds) Learning from Data: Artificial Intelligence and Statistics V, Springer-Verlag, pp 121–130

Chickering DM (2002) Optimal structure identification with greedy search. Journal of Machine Learning Research 3:507–554

Chickering DM, Geiger D, Heckerman D (1995) Learning bayesian networks: Search methods and experimental results. In: Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics, pp 112–128

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

Cowell RG, Dawid AP, Lauritzen S, Spiegelhalter D (2003) Probabilistic Networks and Expert Systems (Information Science and Statistics). Springer

Dash D, Druzdzel MJ (1999) A hybrid anytime algorithm for the construction of causal models from sparse data. In: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI'99), pp 142–149

van Dijk S, van der Gaag LC, Thierens D (2003) A skeleton-based approach to learning bayesian networks from data. In: In proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03), pp 132–143

Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32(200):675–701

Friedman N, Nachman I, Pe'er D (1999) Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99), pp 206–215

Friedman N, Linial M, Nachman I, Pe'er D (2000) Using bayesian network to analyze expression data. Computational Biology 7:601–620

Gámez JA, Puerta JM (2005) Constrained score+(local)search methods for learning bayesian networks. In: 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-05), LNCS vol. 3571, pp 161–173

Geiger D, Heckerman D, King H, Meek C (2001) Stratified exponential families: graphical models and model selection. The Annals of Statistics 29(2):505–529

Haughton DMA (1988) On the choice of a model to fit data from an exponential family. The Annals of Statistics 16(1):342–355

Heckerman D (1997) Bayesian networks for data mining. Data Mining and Knowledge Discovery 1:79–119

Heckerman D, Geiger D, Chickering DM (1995) Learning bayesian networks: The combination of knowledge and statistical data. Machine Learning 20(3):197–243

Holm S (1979) A simple sequential rejective multiple Bonferroni procedures to pairwise multiple comparisons in balanced repeated measures designs. Computational Statistics Quarterly 6:219–231

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

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

Jensen FV, Nielsen TD (2007) Bayesian Networks and Decision Graphs (2ed). Springer Verlag

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

Larrañaga P, Poza M, Yurramendi Y, Murga RH, Kuijpers CMH (1996) Structure learning of bayesian networks by genetic algorithms: A performance analysis of control parameters. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(9):912–926

Margaritis D (2003) Learning bayesian model structure from data. PhD thesis, Carnegie Mellon University

Moral S (2004) An empirical comparison of score measures for independence. In: Proceedings of the 10th IPMU International Conference, pp 1307–1314

Nägele A, Dejori M, Stetter M (2007) Bayesian substructure learning - approximate learning of very large network structures. In: Proceedings of the 18th European conference on Machine Learning (ECML '07), pp 238–249

Neapolitan R (2003) Learning Bayesian Networks. Prentice Hall

Pearl J (1988) Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference. Morgan Kaufmann

Peña JM, Nilsson R, Björkegren J, Tegnér J (2007) Towards scalable and data efficient learning of Markov boundaries. International Journal of Approximate Reasoning 45(2):211–232

Robinson R (1977) Counting unlabeled acyclic digraphs. In: Combinatorial Mathematics, Springer-Verlag, Berlin, vol 622, pp 28–43

Schwarz G (1978) Estimating the dimension of a model. Annals of Statistics 6(2):461–464

Spellman PT, Sherlock G, Zhang MQ, Iyer VR, Anders K, Eisen M, Brown P, Botstein D, Futcher B (1998) Comprehensive identification of cell cycle-regulated genes of the yeast sacccharomyces cerevisiae by microarray hybridization. Molecular Biology of the Cell 9:3273–3297

Spirtes P, Glymour C, Scheines R (1993) Causation, Prediction and Search, Lecture Notes in Statistics, vol 81. Springer Verlag

Statnikov A, Tsamardinos I, CF A (2003) An algorithm for generation of large bayesian networks. Tech. Rep. DSL TR-03-01, Vanderbilt University

Tsamardinos I, Brown LE, Aliferis CF (2006a) The max-min hill-climbing bayesian network structure learning algorithm. Machine Learning 65(1):31–78

Tsamardinos I, Statnikov A, LE B, CF A (2006b) Generating realistic large bayesian networks by tiling. In: Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society FLAIRS Conference, pp 592–597

Verma T, Pearl J (1991) Equivalence and synthesis of causal models. In: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intel-

ligence (UAI'90), Elsevier Science Inc., pp 255–270
WenChen X, Anantha G, Lin X (2008) Improving bayesian network structure learning with mutual information-based node ordering in the k2 algorithm. IEEE Transactions on Knowledge and Data Engineering 20(5):628–640
Witten IH, Frank E (2005) Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). Morgan Kaufmann
Wong ML, Leung KS (2004) An efficient data mining method for learning bayesian networks using an evolutionary algorithm-based hybrid approach. IEEE Trans Evolutionary Computation 8(4):378–404

A Proofs for Proposition 6

In this appendix we prove proposition 6, but firstly we need the following results:

Proposition 7 *Let $\mathcal{G}$ and $\mathcal{G}'$ be two DAGs with $\mathcal{G}_m$ and ${\mathcal{G}'}_m$ the underlying moral graphs of $\mathcal{G}$ and $\mathcal{G}'$ respectively, such that $\mathcal{G}_m \subseteq {\mathcal{G}'}_m$, and for every v-structure in $\mathcal{G}'$ the same v-structure exists in $\mathcal{G}$, then $\mathcal{G}'$ is an I-map of $\mathcal{G}$.*

Proof Assume the contrary, then we have that $\langle X_i, X_j|\mathbf{S}\rangle_{\mathcal{G}'} \wedge \neg \langle X_i, X_j|\mathbf{S}\rangle_{\mathcal{G}}$. Therefore there exists at least one path between X_i and X_j in $\mathcal{G}$ activated by $\mathbf{S}$. We can distinguish the following two cases:

- *Case a) There is one v-structure, $X_a \rightarrow X_c \leftarrow X_b$, in the active path in $\mathcal{G}$. Because the path is* active*, then either the node X_c or (at least) one of its descendants is included in $\mathbf{S}$. Since $X_a \rightarrow X_c \leftarrow X_b$ is in $\mathcal{G}$ then we have that $X_a - X_b$ is present in $\mathcal{G}_m$ and we have as hypothesis that $\mathcal{G}_m \subseteq {\mathcal{G}'}_m$, then the link $X_a - X_b$ is also included in ${\mathcal{G}'}_m$, i.e. the link is in both moral graphs. The presence of the link $X_a - X_b$ in ${\mathcal{G}'}_m$ is due to: (1) the same v-structure $X_a \rightarrow X_c \leftarrow X_b$ is in both graph $\mathcal{G}'$ and $\mathcal{G}$; and/or (2) $X_a - X_b$ exits in $\mathcal{G}'$.*
  In the first case, the subset $\mathbf{S}$ will also activate the path in $\mathcal{G}'$ because the same v-structure is present in both graphs. Furthermore, other variables in the path not belonging to the v-structure will not be present in $\mathbf{S}$ because the path is active in $\mathcal{G}$ so the path is also active in $\mathcal{G}'$ and we therefore get $\neg \langle X_i, X_j|\mathbf{S}\rangle_{\mathcal{G}'}$ which contradicts our hypothesis.
  In the second case, although X_c belongs to $\mathbf{S}$ to activate the path in $\mathcal{G}$, in $\mathcal{G}'$ there will be an active path, the one starting in X_i and ending in X_j sharing the same link in both paths in $\mathcal{G}$ and $\mathcal{G}'$ but we get the link $X_a - X_b$ instead of the subgraph $X_a - X_c - X_b$, as the rest of the nodes in both paths do not belong to $\mathbf{S}$, the path is also active in $\mathcal{G}'$ and we therefore get $\neg \langle X_i, X_j|\mathbf{S}\rangle_{\mathcal{G}'}$ which contradicts our hypothesis.
  If more v-structures are in the path, then the same reasoning can be applied to show that their end nodes will be adjacent in the moral graphs.
- *Case b) There are no v-structures in the path in $\mathcal{G}$. In this case, as the path is active in $\mathcal{G}$, then none of the intermediate nodes can belong to $\mathcal{S}$. We also know that the same path (with possibly different orientations) exists in $\mathcal{G}'$, but in order to block this path in $\mathcal{G}'$ without including any of its intermediate nodes in $\mathbf{S}$, there must be at least one v-structure $X_a \rightarrow X_c \leftarrow X_b$ in the path in $\mathcal{G}'$. $X_a \rightarrow X_c \leftarrow X_b$ is not present in $\mathcal{G}$. However, the nodes X_a and X_b of this v-structure must be adjacent in $\mathcal{G}'$. If not, the same v-structure $X_a \rightarrow X_c \leftarrow X_b$ must be present in $\mathcal{G}$ and this contradicts our assumption of no v-structures in the path in $\mathcal{G}$. As this situation is the same for all the possible v-structures in the path in $\mathcal{G}'$, that is, all of them must have their end nodes connected, then, to block the path the only option is that (at least) one of the intermediate nodes belongs to $\mathbf{S}$, but in this case the path is also blocked in $\mathcal{G}$ which contradicts our hypothesis.*

□

Proposition 8 *Under the same conditions as Proposition 3, with $\mathcal{G}^*$ being the graph that the distribution p was sampled from, 2iCHC (Algorithm 5) returns a graph, $\hat{\mathcal{G}}$, such that ${\mathcal{G}^*}_m \subseteq \hat{\mathcal{G}}_m$, where ${\mathcal{G}^*}_m$ and $\hat{\mathcal{G}}_m$ are the underlying moral graphs of $\mathcal{G}^*$ and $\hat{\mathcal{G}}$ respectively.*

Proof First of all, note that if an adjacency $X_i - X_j$ is in $\mathcal{G}^*$ then it is also in $\hat{\mathcal{G}}$ after the first iteration in the algorithm 2iCHC because there is no subset $\mathbf{S} \subset \mathbf{V}$ such that $I_p(X_i, X_j|\mathbf{S})$. Specifically we have $\neg I_p(X_i, X_j|Pa_{\hat{\mathcal{G}}}(X_i))$, given any subset of parents $Pa_{\hat{\mathcal{G}}}(X_i)$, and so $f_D(X_{p_i}, Pa_{\hat{\mathcal{G}}}(X_{p_i}) \cup \{X_j\}) - f_D(X_{p_i}, Pa_{\hat{\mathcal{G}}}(X_{p_i})) > 0$ for any subset $Pa_{\hat{G}}(X_i)$. As the reverse is also true, that is, $f_D(X_{p_j}, Pa_{\hat{\mathcal{G}}}(X_{p_j}) \cup \{X_i\}) - f_D(X_{p_j}, Pa_{\hat{\mathcal{G}}}(X_{p_j})) > 0$, we can conclude that at some stage of the algorithm, a link between the nodes will be introduced and never subsequently deleted. Therefore, we know that $\hat{\mathcal{G}}$ has at least the same adjacencies as, or maybe more than, $\mathcal{G}^*$.

In addition, we need to prove that for every v-structure $X_a \rightarrow X_c \leftarrow X_b$ in $\mathcal{G}^*$, the same v-structure is in $\hat{\mathcal{G}}$ or, alternatively, that the three nodes involved are totally connected. If 2iCHC correctly discovers the v-structure $X_a \rightarrow X_c \leftarrow X_b$, then there is no problem because there will not be any movement (deletion or reversal) such that this last step can improve the current graph, so the graph $\hat{\mathcal{G}}$ obtained by the algorithm will include the v-structure. In the case of one edge being badly oriented, e.g. $X_a \rightarrow X_c \rightarrow X_b$, the path between X_a and X_b is active, yielding a positive score difference which provokes the inclusion of edge $X_a \rightarrow X_b$, that is, we get a fully-connected subgraph. The only situation in which the algorithm cannot add the edge $X_a \rightarrow X_b$ is if $X_a \in FP(X_b)$ and $X_b \in FP(X_a)$. However, this case is corrected during the second iteration of the algorithm, because (1) FP lists are initialized to be empty, and (2) X_a will never be included in $FP(X_b)$ because X_c opens the path between X_a and X_b as long as it remains as parent of X_b (note that if edge $X_c \rightarrow X_b$ is reversed then we obtain the v-structure).

□

Proposition 9 *Under the same conditions as Proposition 3, with $\mathcal{G}^*$ being the graph from which the distribution p was sampled, 2iCHC (Algorithm 5) returns a graph, $\hat{\mathcal{G}}$, such that every v-structure in $\hat{\mathcal{G}}$ is in $\mathcal{G}^*$ too.*

Proof Let us suppose the contrary, that is, there exists a v-structure, $X_a \rightarrow X_c \leftarrow X_b$, in $\hat{\mathcal{G}}$ but not in $\mathcal{G}^*$. This can be due to:

- *X_a and X_b are adjacent in $\mathcal{G}^*$. However, if this is the case the algorithm will find that X_a and X_b are conditionally dependent given any subset, which gives rise to a positive difference in score when testing addition, and so an edge between X_a and X_b will be added, making X_a and X_b connected in $\hat{\mathcal{G}}$, contradicting in this way our assumption.*

- *X_a and X_b are not adjacent in $\mathcal{G}^*$ but X_c is not a head to head node in the path $X_a - X_c - X_b$. In this case we have $\neg \langle X_a, X_b|\mathbf{S}\rangle_{\mathcal{G}^*}$, for any subset $\mathbf{S}$ not including X_c. However, from this sentence we can deduce that an edge between X_a and X_b must exist in $\hat{\mathcal{G}}$, which contradicts our assumption. The only situation in which the algorithm cannot add the edge $X_a \rightarrow X_b$ is when $X_a \in FP(X_b)$ and $X_b \in FP(X_a)$. However, this case is corrected during the second iteration of the algorithm, because (1) FP lists are initialized to be empty, and (2) X_a will never be included in $FP(X_b)$ because X_c opens the path between X_a and X_b as far as it remains as a parent of X_b (note that if edge $X_c \rightarrow X_b$ is reversed then we obtain the v-structure).*

□

Proposition 6: Under the same conditions as Proposition 3, 2iCHC (Algorithm 5) returns a **minimal I-map**.

Proof: Our first step is to prove that $\hat{\mathcal{G}}$ is an I-map of $\mathcal{G}^*$, $\hat{\mathcal{G}}$ being the result of the execution of the 2iCHC algorithm and $\mathcal{G}^*$ the graph from which distribution p is obtained. The second step is to prove that in fact $\hat{\mathcal{G}}$ is a *minimal* I-map of $\mathcal{G}^*$.

Table 7 BDeu score obtained for each algorithm with the dataset with 500 instances.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-70795	-71636	-71760	-71801	-72196
Alarm1	-5997	-6016	-6016	-6016	-6582
Alarm3	-21482	-21724	-21779	-21785	-21861
Alarm5	-36035	-36261	-36206	-36321	-36479
Barley	-37079	-37079	-37130	-37130	-37079
Child10	-72826	-73045	-73060	-73060	-73290
Child3	-21546	-21546	-21582	-21582	-21762
Child5	-36522	-36596	-36596	-36596	-36789
Child	-6937	-6937	-6937	-6937	-6999
Gene	-255882	-255908	-257890	-258003	-255652
HailFinder10	-286439	-286455	-289754	-289754	-293908
HailFinder3	-86265	-86399	-87476	-87476	-88361
HailFinder5	-142823	-142823	-144823	-144823	-146505
HailFinder	-28998	-28998	-29003	-29003	-29144
Insurance10	-89736	-90007	-90824	-90846	-91284
Insurance3	-27151	-26963	-26995	-27400	-27514
Insurance5	-44500	-44526	-45139	-45177	-45388
Insurance	-8114	-8031	-8031	-8031	-7961
Link	-158683	-159507	-163422	-165361	-173160
Mildew	-31363	-31363	-31363	-31363	-31363
Munin1	-33776	-34131	-34169	-34174	-37749
Pigs	-187597	-188085	-188832	-188916	-187110
Averaged	-76843	-77002	-77672	-77798	-78552

[(1) $\hat{\mathcal{G}}$ is an I-map of $\mathcal{G}^*$] This is straightforward from Propositions 7, 8 and 9.

[(2) minimal condition] Let us suppose the contrary, that is, there exists $X_j \in Pa_{\hat{\mathcal{G}}}(X_i)$ such that $I_p(X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i) \backslash \{X_j\})$. If so, $\hat{\mathcal{G}}$ cannot be a local optimum because there is (at least) one deletion operation with a positive difference.

B Detailed results

Table 8 BDeu score obtained for each algorithm with the dataset with 1000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-133255	-134167	-134296	-134391	-134491
Alarm1	-11187	-11193	-11217	-11217	-11717
Alarm3	-40022	-40053	-40111	-40113	-40573
Alarm5	-67257	-67776	-67987	-68023	-68207
Barley	-69602	-69602	-69700	-69700	-69624
Child10	-138013	-138068	-138177	-138208	-137644
Child3	-41215	-41002	-41002	-41002	-40827
Child5	-69203	-69423	-69430	-69430	-68983
Child	-13198	-13198	-13204	-13204	-13267
Gene	-483310	-483569	-486425	-486603	-482757
HailFinder10	-543039	-544870	-548519	-548519	-560565
HailFinder3	-163723	-163868	-166008	-166008	-168942
HailFinder5	-270552	-271546	-274629	-274629	-279462
HailFinder	-54992	-54909	-54947	-54947	-55673
Insurance10	-168520	-168628	-170218	-170517	-171047
Insurance3	-50882	-50617	-51437	-51539	-51338
Insurance5	-84432	-84832	-85381	-85495	-85549
Insurance	-15198	-15272	-15301	-15301	-15448
Link	-296410	-295548	-298571	-305225	-321158
Mildew	-59655	-59655	-59655	-59655	-59836
Munin1	-60451	-61501	-61540	-61540	
Pigs	-355467	-358150	-358947	-358993	-356404
Averaged	-149006	-149331	-150246	-150606	-152072

Table 9 BDeu score obtained for each algorithm with the dataset with 5000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-609555	-607437	-607678	-608892	-610362
Alarm1	-49980	-50213	-50219	-50219	-50132
Alarm3	-180906	-180904	-181387	-181473	-182189
Alarm5	-306825	-306741	-306991	-307118	-306489
Barley	-300327	-302561	-303342	-303342	-300663
Child10	-648210	-648070	-648120	-648835	-647568
Child3	-192773	-192753	-192770	-193157	-193225
Child5	-325235	-324081	-324081	-324440	-324643
Child	-62363	-62363	-62473	-62473	-62662
Gene	-2241639	-2233011	-2237648	-2238789	-2240738
HailFinder10	-2504094	-2497147	-2554415	-2559254	-2946914
HailFinder3	-750874	-751317	-769487	-769624	-885004
HailFinder5	-1246264	-1247312	-1274504	-1276704	-1471781
HailFinder	-255079	-255370	-255416	-255416	-292635
Insurance10	-758121	-751395	-751500	-762416	-760056
Insurance3	-225984	-223380	-223982	-227145	-229549
Insurance5	-373142	-372508	-373099	-376408	-381313
Insurance	-68891	-68891	-69228	-69228	-69468
Link	-1295509	-1259692	-1277982	-1312438	-1428825
Mildew	-266362	-266362	-266408	-266408	
Munin1	-246392	-248875	-249422	-249422	
Pigs	-1683880	-1683623	-1683623	-1683623	-1690302
Averaged	-703983	-700938	-707397	-710550	-753726

Table 10 Averaged over the five test dataset, the log Likelihood obtained for each model learned with each algorithm with the dataset with 500 instances.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-58714	-59488	-59758	-59871	-60747
Alarm1	-4756	-4776	-4776	-4776	-5346
Alarm3	-17964	-18156	-18285	-18309	-18476
Alarm5	-30025	-30216	-30218	-30341	-30780
Barley	-31716	-31716	-31912	-31912	-31716
Child10	-65323	-65689	-65751	-65751	-66399
Child3	-19455	-19455	-19527	-19527	-19821
Child5	-32679	-32838	-32838	-32838	-33144
Child	-6149	-6149	-6149	-6149	-6289
Gene	-220386	-220531	-224826	-225291	-221126
HailFinder10	-252257	-252324	-258064	-258064	-261249
HailFinder3	-76076	-76176	-77956	-77956	-78402
HailFinder5	-124988	-124988	-128369	-128369	-129525
HailFinder	-25816	-25816	-25831	-25831	-26143
Insurance10	-76060	-76868	-78572	-78623	-78442
Insurance3	-23233	-23088	-23341	-23713	-23724
Insurance5	-37784	-37882	-39030	-39202	-39244
Insurance	-7163	-7135	-7135	-7135	-6937
Link	-132688	-133531	-138340	-142124	-151675
Mildew	-27903	-27903	-27903	-27903	-27903
Munin1	-24583	-24824	-24900	-24907	-28036
Pigs	-165639	-166279	-167447	-167575	-165239
Averaged	-66425	-66629	-67769	-68008	-68653

Table 11 Averaged over the five test dataset, the log Likelihood obtained for each model learned with each algorithm with the dataset with 1000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-117273	-118481	-118646	-118892	-119937
Alarm1	-9364	-9369	-9441	-9441	-10134
Alarm3	-35376	-35461	-35585	-35620	-36410
Alarm5	-59196	-59767	-60117	-60215	-61201
Barley	-61915	-61915	-62123	-62123	-62002
Child10	-126663	-127052	-127232	-127408	-127449
Child3	-38066	-37999	-37999	-37999	-38041
Child5	-63808	-64166	-64275	-64275	-63803
Child	-12137	-12137	-12167	-12167	-12286
Gene	-438589	-439301	-444751	-445325	-439422
HailFinder10	-494937	-495512	-500424	-500424	-514287
HailFinder3	-149470	-149460	-152118	-152118	-154916
HailFinder5	-245387	-246039	-250137	-250137	-255032
HailFinder	-50230	-50174	-50326	-50326	-51218
Insurance10	-148322	-148927	-152741	-153690	-152640
Insurance3	-44372	-44257	-46078	-46471	-45739
Insurance5	-74817	-75671	-77187	-77476	-76435
Insurance	-13744	-13807	-13900	-13900	-14039
Link	-257538	-254138	-257763	-266271	-288860
Mildew	-53122	-53122	-53122	-53122	-54670
Munin1	-46577	-47647	-47931	-47931	
Pigs	-329303	-332145	-333029	-333129	-332002
Averaged	-134458	-134710	-136150	-136692	-138596

Table 12 Averaged over the five test dataset, the log Likelihood obtained for each model learned with each algorithm with the dataset with 5000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	-580619	-578679	-579490	-581386	-587073
Alarm1	-47131	-47062	-47052	-47052	-47630
Alarm3	-172203	-172189	-172705	-172854	-175396
Alarm5	-292838	-292344	-292754	-293000	-294536
Barley	-262499	-264812	-269003	-269003	-261986
Child10	-627823	-628207	-628269	-629260	-630035
Child3	-186729	-187071	-187128	-187651	-188161
Child5	-315020	-314430	-314430	-314929	-315963
Child	-60469	-60469	-60682	-60682	-60943
Gene	-2173853	-2169279	-2174834	-2177324	-2179287
HailFinder10	-2408225	-2399288	-2466006	-2471907	-2903073
HailFinder3	-722999	-723843	-743702	-743975	-872371
HailFinder5	-1201453	-1202873	-1232070	-1234745	-1451475
HailFinder	-246835	-247161	-247306	-247306	-289259
Insurance10	-715716	-708113	-710534	-724879	-719415
Insurance3	-213381	-210549	-211463	-215434	-217505
Insurance5	-350339	-350854	-352362	-357110	-361153
Insurance	-65980	-65980	-66373	-66373	-66809
Link	-1230112	-1193527	-1210959	-1248305	-1389430
Mildew	-246995	-246995	-247085	-247085	
Munin1	-214023	-216687	-216948	-216948	
Pigs	-1651564	-1651568	-1651568	-1651568	-1659522
Averaged	-676289	-673415	-680935	-684737	-733551

Table 13 Structural Hamming distance obtained for each model learned with each algorithm with the dataset with 500 instances.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	552	548	547	548	441
Alarm1	39	43	43	43	37
Alarm3	135	137	137	136	111
Alarm5	255	257	251	253	201
Barley	87	87	87	87	87
Child10	206	232	231	231	206
Child3	78	78	77	77	72
Child5	85	87	87	87	83
Child	22	22	22	22	18
Gene	326	358	429	429	345
HailFinder10	953	955	1009	1009	998
HailFinder3	270	272	293	293	286
HailFinder5	425	423	463	463	445
HailFinder	103	103	103	103	97
Insurance10	395	378	421	416	362
Insurance3	108	92	99	110	98
Insurance5	198	190	211	213	193
Insurance	54	49	49	49	41
Link	1137	1142	1118	1132	909
Mildew	47	47	47	47	47
Munin1	338	327	341	341	301
Pigs	82	91	101	102	60
Averaged	268	269	280	281	247

Table 14 Structural Hamming distance obtained for each model learned with each algorithm with the dataset with 1000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	530	540	536	532	399
Alarm1	43	44	43	43	31
Alarm3	132	129	127	126	95
Alarm5	243	240	232	233	179
Barley	81	81	80	80	80
Child10	152	157	161	154	140
Child3	61	59	59	59	57
Child5	41	65	61	61	50
Child	16	16	20	20	15
Gene	312	318	369	372	283
HailFinder10	852	884	945	945	860
HailFinder3	249	251	282	282	257
HailFinder5	384	399	441	441	386
HailFinder	108	104	105	105	101
Insurance10	386	359	382	387	337
Insurance3	120	104	111	113	89
Insurance5	195	201	211	207	174
Insurance	38	47	46	46	33
Link	1218	1247	1234	1248	833
Mildew	51	51	51	51	50
Munin1	334	316	329	329	
Pigs	131	164	172	171	62
Averaged	254	260	270	270	215

Table 15 Structural Hamming distance obtained for each model learned with each algorithm with the dataset with 5000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	469	460	460	452	325
Alarm1	47	44	44	44	24
Alarm3	123	124	126	125	70
Alarm5	207	212	208	206	139
Barley	64	61	66	66	56
Child10	188	181	182	182	136
Child3	58	63	62	55	50
Child5	65	45	45	53	70
Child	5	5	8	8	12
Gene	289	130	137	140	156
HailFinder10	682	689	799	796	755
HailFinder3	197	193	223	222	211
HailFinder5	328	321	357	355	343
HailFinder	112	113	111	111	88
Insurance10	398	354	341	382	302
Insurance3	107	95	100	105	88
Insurance5	175	156	165	195	166
Insurance	29	29	31	31	31
Link	1153	1067	1045	1067	687
Mildew	49	49	48	48	
Munin1	311	327	329	329	
Pigs	73	67	67	67	38
Averaged	238	220	229	233	187

Table 16 Statistics for each algorithm with the dataset with 500 instances. For the MMHC algorithm the statistics computed in each one of its two phases are given.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	309332	230598	149119	137747	156245
Alarm1	2817	2277	1740	1708	2619
Alarm3	26376	20379	12997	12206	15688
Alarm5	76010	59611	38530	34113	41174
Barley	3452	2987	1933	1933	3006
Child10	76366	61759	38693	36585	48743
Child3	6600	5412	3565	3422	5396
Child5	18505	14659	9502	9121	13339
Child	758	575	462	462	942
Gene	1440236	1023008	647320	615489	758327
HailFinder10	594616	464697	271157	269168	361867
HailFinder3	51916	41441	24058	23801	36045
HailFinder5	148351	116180	67179	66889	94804
HailFinder	5764	4485	2912	2849	5071
Insurance10	170540	122753	77126	72016	88168
Insurance3	15123	11381	7440	6749	10244
Insurance5	41319	31214	19272	17976	24695
Insurance	1468	1051	911	888	1609
Link	1064942	909166	674669	627014	471364
Mildew	1654	1474	889	889	1165
Munin1	63035	56491	45947	44955	49456
Pigs	491364	293247	181527	171782	262969
Averaged	209570	157948	103498	98080	111497

Table 17 Statistics for each algorithm with the dataset with 1000 instances. For the MMHC algorithm the statistics computed in each one of its two phases are given. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	323346	230726	149751	135060	159744
Alarm1	3008	2503	1831	1747	2864
Alarm3	28161	20933	13518	12282	16249
Alarm5	78811	58157	37526	33969	42033
Barley	3722	3162	2131	2098	3606
Child10	85773	64342	41347	37702	50736
Child3	7550	5616	3880	3680	6190
Child5	20921	15590	10266	9647	14748
Child	848	679	501	490	1322
Gene	1479383	1001751	654086	623669	784846
HailFinder10	631821	482154	282450	278356	364335
HailFinder3	56843	44367	25202	24819	37362
HailFinder5	158283	120718	71284	70071	96397
HailFinder	6035	4643	2974	2959	5168
Insurance10	180862	126856	79040	73208	95584
Insurance3	16016	11548	7524	6920	12381
Insurance5	44668	30458	19132	18555	28761
Insurance	1541	1205	945	930	2110
Link	1184015	939053	687314	623712	584523
Mildew	1885	1629	1063	1063	1520
Munin1	65786	57660	48117	47132	
Pigs	545696	321204	191470	177135	431766
Averaged	231390	166062	108725	101813	130583

Table 18 Statistics for each algorithm with the dataset with 5000 instances. For the MMHC algorithm the statistics computed in each one of its two phases are given. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	355413	227383	151603	137904	168698
Alarm1	3155	2549	1916	1840	3695
Alarm3	28861	19817	13143	12559	18125
Alarm5	86067	55164	36592	34141	45279
Barley	4878	3490	2525	2338	6006
Child10	103637	67296	43795	41075	58544
Child3	9394	6423	4366	4156	8754
Child5	25869	16702	11456	10732	18459
Child	899	732	581	572	2038
Gene	1543025	989291	672788	642822	876744
HailFinder10	748072	578112	312446	293358	376455
HailFinder3	64328	49437	27133	26341	40643
HailFinder5	181413	137357	73837	71950	101619
HailFinder	6849	4919	3336	3132	5556
Insurance10	218216	150475	89424	73964	126251
Insurance3	18468	13455	8652	7440	21245
Insurance5	53677	37501	23311	19284	45436
Insurance	1791	1371	1130	1067	4306
Link	1263499	959893	686947	610936	833068
Mildew	2207	1828	1267	1236	
Munin1	73088	62697	54150	52519	
Pigs	562130	284323	184224	181202	12582998
Averaged	263982	180285	117460	108841	767196

Table 19 Number of links in each model learned by each algorithm with the dataset with 500 instances.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	455	458	443	438	331
Alarm1	42	42	42	42	32
Alarm3	125	128	122	121	99
Alarm5	221	223	216	215	172
Barley	25	25	24	24	25
Child10	182	181	179	179	152
Child3	52	52	50	50	41
Child5	87	85	85	85	75
Child	20	20	20	20	16
Gene	928	924	874	868	902
HailFinder10	480	478	430	430	385
HailFinder3	138	138	124	124	115
HailFinder5	243	243	218	218	200
HailFinder	48	48	47	47	39
Insurance10	328	322	306	305	292
Insurance3	96	95	91	90	86
Insurance5	161	161	150	148	140
Insurance	27	26	26	26	25
Link	756	751	712	697	442
Mildew	13	13	13	13	13
Munin1	148	152	149	149	95
Pigs	592	590	581	580	589
Averaged	235	234	223	221	194

Table 20 Number of links in each model learned by each algorithm with the dataset with 1000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	485	478	476	471	376
Alarm1	48	48	46	46	38
Alarm3	139	136	132	131	106
Alarm5	237	235	227	226	188
Barley	31	31	30	30	30
Child10	221	217	214	212	189
Child3	62	61	61	61	56
Child5	107	104	103	103	98
Child	22	22	21	21	20
Gene	969	968	933	930	947
HailFinder10	539	539	506	506	430
HailFinder3	160	160	145	145	127
HailFinder5	275	275	251	251	220
HailFinder	54	53	50	50	43
Insurance10	370	368	350	345	331
Insurance3	116	115	105	103	101
Insurance5	181	179	170	168	163
Insurance	32	32	31	31	27
Link	913	920	900	881	551
Mildew	20	20	20	20	18
Munin1	162	160	158	158	
Pigs	625	630	629	628	592
Averaged	267	266	257	255	221

Table 21 Number of links in each model learned by each algorithm with the dataset with 5000 instances. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	567	558	543	534	436
Alarm1	53	54	54	54	44
Alarm3	150	151	149	148	124
Alarm5	265	267	260	258	211
Barley	49	49	49	49	48
Child10	274	268	269	267	237
Child3	82	80	79	78	70
Child5	135	132	132	131	119
Child	27	27	25	25	25
Gene	1037	1009	1003	998	975
HailFinder10	699	711	639	634	406
HailFinder3	202	200	182	181	116
HailFinder5	329	328	304	302	186
HailFinder	64	63	61	61	34
Insurance10	475	480	475	451	435
Insurance3	140	141	138	134	130
Insurance5	241	241	234	223	215
Insurance	41	41	40	40	36
Link	982	994	976	959	590
Mildew	30	30	29	29	
Munin1	204	204	200	200	
Pigs	613	611	611	611	592
Averaged	321	320	311	307	251

Table 22 Ratio between the gain obtained by each model learned with the dataset with 500 instances against the empty model and the number of statistics computed.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	0.12	0.16	0.25	0.27	0.23
Alarm1	2.03	2.51	3.28	3.34	1.96
Alarm3	0.42	0.54	0.83	0.88	0.68
Alarm5	0.25	0.32	0.49	0.55	0.45
Barley	2.42	2.79	4.22	4.22	2.78
Child10	0.28	0.34	0.54	0.57	0.41
Child3	0.97	1.18	1.77	1.84	1.11
Child5	0.59	0.74	1.14	1.19	0.79
Child	2.98	3.93	4.9	4.9	2.25
Gene	0.06	0.09	0.14	0.14	0.12
HailFinder10	0.15	0.2	0.31	0.32	0.23
HailFinder3	0.52	0.65	1.04	1.05	0.68
HailFinder5	0.32	0.4	0.65	0.65	0.45
HailFinder	1.53	1.96	3.02	3.08	1.67
Insurance10	0.23	0.31	0.47	0.51	0.42
Insurance3	0.75	1.01	1.51	1.61	1.06
Insurance5	0.47	0.62	0.95	1	0.73
Insurance	2.36	3.33	3.84	3.94	2.3
Link	0.06	0.07	0.09	0.09	0.1
Mildew	1.65	1.85	3.07	3.07	2.34
Munin1	0.38	0.42	0.52	0.53	0.42
Pigs	0.13	0.21	0.34	0.36	0.24
Averaged	0.85	1.07	1.52	1.55	0.97

Table 23 Ratio between the gain obtained by each model learned with the dataset with 1000 instances against the empty model and the number of statistics computed. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	0.24	0.33	0.5	0.56	0.47
Alarm1	3.93	4.72	6.41	6.72	3.86
Alarm3	0.79	1.06	1.63	1.79	1.3
Alarm5	0.49	0.66	1.01	1.11	0.88
Barley	4.95	5.82	8.54	8.68	5.08
Child10	0.54	0.71	1.1	1.21	0.9
Child3	1.79	2.42	3.51	3.7	2.19
Child5	1.11	1.47	2.22	2.36	1.58
Child	5.68	7.09	9.55	9.76	3.53
Gene	0.13	0.19	0.28	0.29	0.24
HailFinder10	0.3	0.4	0.66	0.67	0.48
HailFinder3	1	1.28	2.15	2.19	1.38
HailFinder5	0.62	0.81	1.31	1.33	0.92
HailFinder	3.17	4.13	6.39	6.43	3.51
Insurance10	0.45	0.64	0.98	1.04	0.81
Insurance3	1.55	2.15	3.06	3.27	1.89
Insurance5	0.89	1.28	1.95	2	1.33
Insurance	5	6.35	8	8.13	3.52
Link	0.12	0.16	0.21	0.22	0.19
Mildew	4.52	5.23	8.02	8.02	4.59
Munin1	0.8	0.89	1.07	1.09	
Pigs	0.24	0.39	0.65	0.71	0.29
Averaged	1.79	2.25	3.25	3.34	1.85

Table 24 Ratio between the gain obtained by each model learned with the dataset with 5000 instances against the empty model and the number of statistics computed. An empty cell means that the algorithm is not able to deal with the given dataset.

	HC	CHC*	iCHC	2iCHC	MMHC
Alarm10	1.11	1.75	2.62	2.87	2.31
Alarm1	18.93	23.46	31.21	32.50	16.03
Alarm3	3.92	5.71	8.57	8.95	6.06
Alarm5	2.32	3.62	5.45	5.84	4.37
Barley	28.78	39.57	53.03	57.27	23.46
Child10	2.30	3.54	5.44	5.77	4.04
Child3	7.78	11.33	16.65	17.36	8.19
Child5	4.62	7.19	10.48	11.14	6.42
Child	27.87	34.23	42.76	43.43	12.06
Gene	0.62	0.98	1.43	1.49	1.09
HailFinder10	1.38	1.80	3.12	3.31	1.43
HailFinder3	4.82	6.26	10.67	10.98	3.96
HailFinder5	2.86	3.77	6.62	6.76	2.65
HailFinder	14.79	20.52	30.22	32.19	10.59
Insurance10	1.99	2.94	4.91	5.75	3.41
Insurance3	7.13	10.00	15.45	17.43	6.01
Insurance5	4.17	5.95	9.51	11.25	4.69
Insurance	23.16	30.26	36.36	38.51	9.44
Link	0.62	0.86	1.17	1.25	0.75
Mildew	28.37	34.25	49.34	50.58	
Munin1	3.99	4.61	5.33	5.49	
Pigs	1.14	2.26	3.49	3.54	0.05
Averaged	8.02	10.80	14.96	15.88	6.35