One iteration CHC algorithm for learning Bayesian networks: an effective and efficient algorithm for high dimensional problems

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

Received: 18 September 2012 / Accepted: 1 October 2012 / Published online: 18 October 2012
© Springer-Verlag Berlin Heidelberg 2012

Abstract It is well known that learning Bayesian networks from data is an NP-hard problem. For this reason, usually metaheuristics or approximate algorithms have been used to provide a good solution. In particular, the family of hill climbing algorithms has a key role in this scenario because of its good trade-off between computational demand and the quality of the learned models. In addition, these algorithms have several good theoretical properties. In spite of these characteristics of quality and efficiency, when it comes to dealing with high-dimensional datasets, they can be improved upon, and this is the goal of this paper. Recent papers have tackled this problem, usually by dividing the learning task into two or more iterations or phases. The first phase aims to constrain the search space, and, once the space is pruned, the second one consists of a (local) search in this constrained search space. Normally, the first iteration is the one with the highest computational complexity. One such algorithm is constrained hill climbing (CHC), which in its initial iteration not only progressively constrains the search space, but also learns good quality Bayesian networks. A second iteration, or even more, is used in order to improve these networks and also to ensure the good theoretical properties exhibited by the classical hill climbing algorithm. In this latter algorithm we can see that the first iteration is extremely fast when compared to simi-

lar algorithms, but the performance decays over the rest of the iterations with respect to the saved CPU time. In this paper, we present an improvement on this CHC algorithm, in which, to put it, briefly, we avoid the last iteration while still obtaining the same theoretical properties. Furthermore, we experimentally test the proposed algorithms over a set of different domains, some of them quite large (more than 1,000 variables), in order to study their behavior in practice.

Keywords Bayesian networks · Machine learning · Score-based learning · Local search · Scalability

1 Introduction

The goal of data mining can be understood as compressing the available data into a more compact representation called a *model*. Later, this model can be used to tackle different descriptive (e.g. identifying dependences relations, clusters, etc.) or predictive (e.g. classification, computing posterior beliefs) tasks. *Bayesian Networks* [24,27,31] have become one of the favorite knowledge representation formalisms for model-based data mining because of their double descriptive/predictive capability and their innate uncertainty management.

Bayesian Networks (BNs) are graphical models that are able to efficiently represent and manipulate n -dimensional probability distributions [31]. The knowledge base that a BN encodes can be viewed as a double representation model divided into a qualitative (a directed acyclic graph or DAG) and a quantitative part (a set of locally specified probability distributions). Thus, descriptive tasks are carried out by performing *relevance analysis* over the graph, while predictive tasks are based on a clever use of the (in)dependences codified in the DAG to allow efficient probabilistic inference. As BNs are such attractive models, and due to the

J. A. Gámez (✉) · J. M. Puerta
Department of Computing Systems, Intelligent Systems and Data Mining Group-i3A, University of Castilla-La Mancha,
02071 Albacete, Spain
e-mail: jose.gamez@uclm.es

J. M. Puerta
e-mail: jose.puerta@uclm.es

J. L. Mateo
Centre for Organismal Studies, Im Neuenheimer, Feld 230,
69120 Heidelberg, Germany
e-mail: juan.mateo@cos.uni-heidelberg.de

increasing availability of data, it is not strange to find such a large number of works in the literature that tackle the BN structure learning problem. Somewhat generalizing, there are two main approaches for learning BNs:

- *Score+search methods* A function f is used to score a network/DAG with respect to the training data, and a search method is used to look for the network with the best score. Different scoring metrics [11, 25, 30] and search methods, mainly of a heuristic (e.g. [3, 4, 9, 14, 16, 25, 30]) and metaheuristic (e.g. [2, 8, 12, 28]) nature have been proposed due to the NP-hardness of the BN structure learning problem [7].
- *Constraint-based methods* The idea underlying these methods is to satisfy as many independences present in the data as possible [30, 35]. Statistical hypotheses testing is used to determine the validity of conditional independence sentences. There also exist hybrid algorithms that combine these two approaches, e.g. [1] or even hybrid scoring metrics [11].

Dealing with larger data sets, and therefore designing scalable algorithms, is nowadays one of the major challenges of machine learning [21]. In BN structure learning in the space of DAGs, local search-based approach (e.g. [9, 13, 14, 16, 17, 25, 30, 36]) stands out when dealing with large datasets (very large number of variables), because of its good trade-off between the resources required (e.g. CPU time) and the accuracy of the model obtained.

Our main motivation in this paper is to scale up local search algorithms for learning BNs, such as hill climbing, while maintaining the theoretical properties that this method offers. As the cardinality of this search space is super-exponential [33], a good idea, especially in domains with a large number of variables, is to limit in the areas of the search space to be visited in some way. This idea is not new and has been exploited previously in the literature, as we will briefly review in Sect. 2.2.

Our proposal in this paper is to develop a *one-stage single-iteration* constrained hill climbing algorithm, with the goal of significantly reducing its running time and so applying it to databases with a larger number of variables. Our experiments confirm that the resulting hill climbing algorithm, called the *FastCHC algorithm*, is faster than previous approaches from the state-of-the-art algorithms, while maintaining the quality of the discovered network close to that of those learnt by this family of algorithms.

This paper is structured as follows: We begin in Sect. 2 by presenting the notation and definitions concerning BNs necessary to build our proposal. In Sect. 3, we describe in detail some local search methods for learning BNs and previous algorithms in the CHC family. Section 4 is devoted to explaining in our proposal. In Sect. 5 we describe the experi-

ments carried out to validate our claim about the quality of our new algorithm. Finally, in Sect. 6 we present our conclusions.

2 Preliminaries

2.1 Bayesian networks

Bayesian networks (BNs) are graphical models that can efficiently represent and manipulate n -dimensional probability distributions [31]. This representation has two components that, respectively, codify qualitative and quantitative knowledge:

- A graphical structure, or more precisely a DAG, $\mathcal{G} = (V, E)$, where the nodes in $V = \{X_1, X_2, \dots, X_n\}$ represent the random variables¹ from the problem we are modeling, and the topology of the graph (the arcs in $E \subseteq V \times V$) 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 (Θ), usually conditional probability tables, drawn from the graph structure: For each variable $X_i \in V$ we have a conditional probability distribution $P(X_i | \mathbf{pa}(X_i))$, where $\mathbf{pa}(X_i)$ represents any combination of the values of the variables in $\mathbf{Pa}(X_i)$, and $\mathbf{Pa}(X_i)$ is the parent set² of X_i in $\mathcal{G}$. From these conditional distributions we can recover the joint probability distribution over V thanks to the Markov Condition:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | \mathbf{Pa}_{\mathcal{G}}(X_i)) \quad (1)$$

This decomposition of the joint distribution gives rise to important savings in storage requirements and also allows the definition of efficient probabilistic inference algorithms by means of local propagation schemes [27].

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

Definition 1 A node or variable X is a *collider* in a path π if X has two incoming edges, i.e. we have the subgraph $A \rightarrow X \leftarrow B$ (also known as a head to head node). If the tail nodes (A and B) of a collider node are not adjacent in $\mathcal{G}$, this subgraph is called a *v-structure* in X .

¹ We use standard notation, i.e., 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 $\mathbf{Pa}_{\mathcal{G}}(X_i)$ to clarify the notation.

Definition 2 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 3 Two nodes X and Y are *d-separated* by $\mathbf{Z}$ in graph G if and only if every path from X to Y is blocked by $\mathbf{Z}$. Two nodes are *d-connected* if they are not *d-separated*.

Definition 4 A DAG $\mathcal{G}$ is an *I-map* of a probability distribution p if $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}} \implies I_p(\mathbf{X}, \mathbf{Y} | \mathbf{Z})$. It 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 (i.e. $\mathcal{G}$ is a *perfect-map* of p) or we will say that p and $\mathcal{G}$ are *faithful* to each other [30,35].

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 [31].

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 in p and $\mathcal{G}$.

2.2 Related work

The idea of learning BNs in large domains by constraining the search space is not new and has been exploited previously. For example, in [5] they restrict the number of possible parents for a variable to be the k most correlated with it, and then, the K2 algorithm [10] is used to learn the BN structure. By using no restriction on the order of the nodes, we can use the Max–Min Hill Climbing (MMHC) algorithm [36], which 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 one 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 [29], the first step is carried out as in MMHC but in the second phase substructures are learned using the information gathered in first phase. In [16], 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.

There also exists global search methods in which the search space is also restricted, for example in [37] 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 this skeleton. The approach taken in [39] is slightly different in that it also uses a first stage based on low order conditional independence tests.

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. In [17], the authors propose a different way of learning BNs, namely by carrying out these two stages simultaneously. Thus, a hill climbing algorithm is launched directly without previously restricting the search space, and then it takes 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. However, in order to maintain the nice theoretical property that under the faithfulness condition the CHC algorithm always returns a *minimal I-map*, at least one extra iteration should be executed. These last iterations are usually short in local steps but are time-consuming in global terms due to the extra load of computing new statistics from data. This means that the time saving with respect to previous two-stage proposals diminishes, even though the first iteration is really very fast.

Our proposal in this paper is to avoid these extra iterations in order to improve the CHC algorithm significantly reducing its running time and so applying it to databases with a larger number of variables. To do this, we propose to check the restrictions applied previously for each variable parent set in order to guarantee the minimal I-map condition in only one iteration.

3 Learning BNs by local search methods

In the score+search approach, the problem of learning the structure of a Bayesian network can be stated as follows: given a training dataset $D = \{\mathbf{v}^1, \dots, \mathbf{v}^m\}$ of instances (configurations of values) of $\mathbf{V}$, find the DAG $\mathcal{G}^*$ such that

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

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 local steps. At each local step, the algorithm only considers minimum or 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 most times maximizes f locally rather than globally. Different strategies are used to try to escape from local optima: restarts, randomness, etc.

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 score, i.e. the subgraphs formed by a node and its parents in $\mathcal{G}$. Formally, if f is decomposable then:

$$f(\mathcal{G} : D) = \sum_{i=1}^n f_D(X_i, \text{Pa}_{\mathcal{G}}(X_i)) \quad (3)$$

$$f_D(X_i, \text{Pa}_{\mathcal{G}}(X_i)) = f_D(X_i, \text{Pa}_{\mathcal{G}}(X_i) : N_{X_i, \text{Pa}_{\mathcal{G}}(X_i)}) \quad (4)$$

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

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 method 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. If we use an HC algorithm, we will have to measure the following differences when evaluating the improvement obtained by a neighbor DAG:

1. Addition of $X_j \rightarrow X_i$: $f_D(X_i, \text{Pa}(X_i) \cup \{X_j\}) - f_D(X_i, \text{Pa}(X_i))$.
2. Deletion of $X_j \rightarrow X_i$: $f_D(X_i, \text{Pa}(X_i) \setminus \{X_j\}) - f_D(X_i, \text{Pa}(X_i))$.
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 $[f_D(X_i, \text{Pa}(X_i) \setminus \{X_j\}) - f_D(X_i, \text{Pa}(X_i))] + [f_D(X_j, \text{Pa}(X_j) \cup \{X_i\}) - f_D(X_j, \text{Pa}(X_j))]$

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

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 for the first time, the obtained value $f_D(\cdot)$ is added to a cache. Subsequent computations can be avoided by just checking that cache.

```

Input:  $D$ : A dataset defined over variables  $\mathbf{V} = \{X_1, \dots, X_n\}$ 
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  $\mathcal{B}$ 

1  $\mathcal{G} \leftarrow \mathcal{G}_0$ ;
2  $f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$ ;           //  $f$ : decomposable scoring metric
3 improvement  $\leftarrow$  true;
4 while improvement do
5   improvement  $\leftarrow$  false;
   //neighbors generated by addition
6   For each node  $X_i$  and each node  $X_j \notin \text{Pa}_{\mathcal{G}}(X_i)$  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
7   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$ , 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
8   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$  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
9   Let  $d^* = \max_{k=a,d,r} diff_k$  and  $move^*$  its corresponding change;
10  if  $d^* > 0$  then
11    improvement  $\leftarrow$  true;
12     $\mathcal{G} \leftarrow$  apply  $move^*$  over  $\mathcal{G}$ ;
13     $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}} + d^*$ ;
14  end
15 end
16 return  $\mathcal{G}$ ;

```

Algorithm 1: Structural learning of BNs by using a Hill Climbing (HC) algorithm

3.1 Theoretical considerations

In this section, we review some (desirable) properties of scoring metrics, most of which are taken from [9]. These concepts will constitute the theoretical basis of our proposal.

Definition 5 A scoring metric f is *score equivalent* if for any pair of equivalent DAGs, $\mathcal{G}$ and $\mathcal{G}'$, $f(\mathcal{G} : D) = f(\mathcal{G}' : D)$.

Two DAGs are equivalent if they lead to the same essential graph, that is, if they share the same skeleton and the same v-structures [38].

Definition 6 (*Consistent scoring criterion* [9]) Let D be a dataset containing m -independent and identically distributed (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 $\mathcal{H}$ (has less parameters), then $f(\mathcal{G} : D) > f(\mathcal{H} : D)$.

A probability distribution p is contained in a DAG $\mathcal{G}$ if there exists a set of parameter values Θ such that the Bayesian network defined by $(\mathcal{G}, \Theta)$ represents p exactly. Of course, if two graphs are correct, then the sparser one should receive more merit.

Proposition 1 (From [9,22,23]) *The Bayesian scoring criterion is score equivalent and consistent.*

Definition 7 (*Locally Consistent scoring criterion* [9]) 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 $\neg I_p(X_i, X_j | \text{Pa}_{\mathcal{G}}(X_j))$, then $f(\mathcal{G} : D) < f(\mathcal{G}' : D)$.
2. If $I_p(X_i, X_j | \text{Pa}_{\mathcal{G}}(X_j))$, then $f(\mathcal{G} : D) > f(\mathcal{G}' : D)$.

This is the main result for the proposal in [17], and the extension/improvement we propose in this paper, 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 isomorphic³ to a graph.

Proposition 2 [9] *The Bayesian scoring criterion is locally consistent.*

Particular Bayesian scores for which Propositions 1 and 2 hold are BDe (Bayesian Dirichlet score with the assumption of likelihood equivalence) [25], and BIC (Bayesian Information Criterion) [34].

³ In fact, [9] proves that the isomorphic condition can be relaxed.

3.2 Constrained hill climbing methods

The hill climbing (HC) algorithm with {arc-addition, arc-deletion, arc-reversal} operations is without any doubt the most frequently used algorithm because of its ease of implementation, efficiency and the quality of the output it offers. That quality is supported by the fact that this algorithm asymptotically guarantees a minimal I-map [17, Proposition 3].

```

Input:  $D$ : A dataset defined over variables  $\mathbf{V} = \{X_1, \dots, X_n\}$ 
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  $\mathcal{B}$ 

1  $\mathcal{G} \leftarrow \mathcal{G}_0$ ;
2  $f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$ ; //: decomposable scoring metric
3 For each  $X_i$  do  $FP(X_i) = \emptyset$ ;
4 improvement  $\leftarrow$  true;
5 while improvement do
6   improvement  $\leftarrow$  false;
   //neighbors generated by addition
7   For each node  $X_i$  and each node  $X_j \notin (\text{Pa}_{\mathcal{G}}(X_i) \cup FP(X_i))$  such
   that  $X_j \rightarrow X_i$  does not introduce a directed cycle in  $\mathcal{G}$ , compute the
   difference  $diff = f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . If  $diff < 0$  then add
    $\{X_j\}$  to  $FP(X_i)$  and add  $\{X_i\}$  to  $FP(X_j)$ . Store the change which
   maximizes  $diff$  in  $\langle change_a, diff_a \rangle$ ;
   //neighbors generated by deletion
8   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$ , compute the
   difference  $diff = f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . If  $diff > 0$  then add
    $\{X_j\}$  to  $FP(X_i)$  and add  $\{X_i\}$  to  $FP(X_j)$ . Store the change which
   maximizes  $diff$  in  $\langle change_d, diff_d \rangle$ ;
   //neighbors generated by reversal
9   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$  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
    $f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . 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  $\langle change_r, diff_r \rangle$ ;
   //Checking if improvement
10  Let  $d^* = \max_{k=a,d,r} diff_k$  and  $move^*$  its corresponding change;
11  if  $d^* > 0$  then
12    improvement  $\leftarrow$  true;
13     $\mathcal{G} \leftarrow$  apply  $move^*$  over  $\mathcal{G}$ ;
14     $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}} + d^*$ ;
15  end
16 end
17 return  $\mathcal{G}$ ;

```

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

In [17], a *constrained* hill climbing (CHC) algorithm is proposed. The scheme of this proposal is shown in Algorithm 2, and as can be observed, it is almost identical to HC (see Algorithm 1), the only difference being the inclusion of the *forbidden parent sets*, $FP(\cdot)$. For each variable,

an $FP()$ set is considered and updated in each step of the greedy search. Thus, the nodes included in $FP(X)$ are not taken into account as possible parents for X during the rest of the search. The CHC algorithm, as well as HC, assures *monotonicity*, i.e., in each step an improvement is guaranteed and CHC, like HC, stops when there is no neighbour of $\mathcal{G}$ which improves $f(\mathcal{G})$, therefore *termination* is also guaranteed. However, CHC cannot ensure that $\hat{\mathcal{G}} = CHC(\mathcal{G}_0)$ is, asymptotically, an I-map of p , as HC does. To solve these problems, The CHC* algorithm is proposed in [17, 19], where the output of CHC is used as the input for unconstrained hill climbing. This latter algorithm shares the property of HC that the resulting DAG is a minimal I-map.

Two other successful variants of CHC*, called iCHC and 2iCHC, respectively, are also introduced in [17]. 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, the algorithm could escape from the current point as an unconstrained HC will do. 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 cleared at the beginning, we can be sure that CHC will also stop without making any modification, and so we get a minimal I-map. Algorithm 2iCHC just carries out two iterations, which it is proven to be enough to ensure that a minimal I-map is obtained [17, Proposition 4]. Obviously, this second approach is faster than the first one, though sometimes it obtains networks whose score is slightly worse.

4 The one iteration CHC Algorithm: FastCHC

The necessity of iterating the CHC algorithm is due to the possibility that the arcs belonging to a v-structure in the true graph are discovered in the wrong direction during the learning process. If this is the case, an I-map can be recovered in two different ways. First, those arcs can be reversed, but in practice this is difficult to do, and there even exists situations in which the reversal operation for those arcs is impossible in the context of a local search over the current graph. The second solution is to add an arc between the tail nodes of the v-structure, however, this action is not easy to do in the CHC approach. The reason comes from the fact that both tail nodes ($T1$ and $T2$) are marginally independent given a subset of nodes S in the original (true) distribution, and therefore, when the algorithm checks the link between $T1$ and $T2$ with the subset S as the parent set of $T1$ or $T2$, $T1$ will be included in $FP(T2)$ and/or vice versa during that stage of the search. This is the reason for the success of iterated versions of CHC, because when starting a new iteration, the $FP()$ sets are empty and so that type of arcs can be reversed.

In this paper our challenge, and main contribution, is to be able to reverse those arcs but in a single-iteration CHC algo-

rithm. The first consequence would be a neater algorithm, and secondly, hopefully, a more efficient one than those iterated, given that only one iteration is performed. Of course, as the nodes in the wrongly discovered v-structure would be totally connected, the output of this new algorithm will still be a minimal I-map.

```

Input:  $D$ : A dataset defined over variables  $\mathbf{V} = \{X_1, \dots, X_n\}$ 
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  $B$ 

1  $\mathcal{G} \leftarrow \mathcal{G}_0$ ;
2  $f_{\mathcal{G}} \leftarrow f(\mathcal{G} : D)$ ; //  $f$ : decomposable scoring metric
3 For each  $X_i$  do  $FP(X_i) = \emptyset$ ;
4 improvement  $\leftarrow$  true;
5 while improvement do
6   improvement  $\leftarrow$  false;
   //neighbors generated by addition
7   For each node  $X_i$  and each node  $X_j \notin (\text{Pa}_{\mathcal{G}}(X_i) \cup FP(X_i))$  such
   that  $X_j \rightarrow X_i$  does not introduce a directed cycle in  $\mathcal{G}$ , compute the
   difference  $diff = f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . If  $diff < 0$  then add
    $\{X_j\}$  to  $FP(X_i)$  and add  $\{X_i\}$  to  $FP(X_j)$ . Store the change which
   maximizes  $diff$  in  $\langle change_a, diff_a \rangle$ ;
   //neighbors generated by deletion
8   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$ , compute the
   difference  $diff = f(\mathcal{G} - \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . If  $diff > 0$  then add
    $\{X_j\}$  to  $FP(X_i)$  and add  $\{X_i\}$  to  $FP(X_j)$ . Store the change which
   maximizes  $diff$  in  $\langle change_d, diff_d \rangle$ ;
   //neighbors generated by reversal
9   For each node  $X_i$  and each node  $X_j \in \text{Pa}_{\mathcal{G}}(X_i)$  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
    $f(\mathcal{G} + \{X_j \rightarrow X_i\} : D) - f_{\mathcal{G}}$ . 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  $\langle change_r, diff_r \rangle$ ;
   //Checking if improvement
10  Let  $d^* = \max_{k=a,d,r} diff_k$  and  $move^*$  its corresponding change;
11  if  $d^* > 0$  then
12    improvement  $\leftarrow$  true;
13     $\mathcal{G} \leftarrow$  apply  $move^*$  over  $\mathcal{G}$ ;
14     $f_{\mathcal{G}} \leftarrow f_{\mathcal{G}} + d^*$ ;
15    if  $move^*$  is adding  $X_j \rightarrow X_i$  then
16       $FP(X_i) = FP(X_i) \setminus \text{NB}_j$ 
17      For all  $X_a \in \text{NB}_j$  do  $FP(X_a) = FP(X_a) \setminus \{X_i\}$ 
18       $FP(X_j) = FP(X_j) \setminus \text{NB}_i$ 
19      For all  $X_b \in \text{NB}_j$  do  $FP(X_b) = FP(X_b) \setminus \{X_j\}$ 
20    end
21  end
22 end
23 return  $\mathcal{G}$ ;

```

Algorithm 3: Structural learning of BNs by using a Fast Constrained Hill Climbing (FastCHC) algorithm

The proposed method, called FastCHC, is described in Algorithm 3, and is basically the same code as in CHC (see Algorithm 2). The difference with respect to the CHC

Fig. 1 Graphs for the Example 1: **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**

algorithm is in lines 15–20. The purpose of these is to allow the possible adding of dependences in order to satisfy the I-map condition when a v-structure is not properly identified. Specifically, when addition of $X_j \rightarrow X_i$ is the selected operation, then any node adjacent to X_j is removed from $FP(X_i)$ and any node adjacent to X_i is removed from $FP(X_j)$. By updating the FP sets in this way, in subsequent operations it is possible to add an arc from any neighbor of X_i to X_j and vice versa. Furthermore, the opposite direction ($X_i \rightarrow X_j$) is also considered, removing X_i from $FP(X_a)$ and X_j from $FP(X_b)$. The following example illustrates this behaviour.

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

- (1) Add $A \rightarrow B$: $diff \gg 0$ because $A \rightarrow B \in \mathcal{G}_t$, so $\neg I(A, B|\emptyset)_{\mathcal{G}_t}$.
- (2) Add $A \rightarrow C$: $diff < 0$ because $I(A, C|\emptyset)_{\mathcal{G}_t}$: $FP(C) = \{A\}$, $FP(A) = \{C\}$.
- (3) Add $A \rightarrow D$: $diff < 0$ because $I(A, D|\emptyset)_{\mathcal{G}_t}$: $FP(D) = \{A\}$, $FP(A) = \{C, D\}$.
- (4) Add $B \rightarrow A$: $diff \gg 0$ because $A \rightarrow B \in \mathcal{G}_t$, so $\neg I(A, B|\emptyset)_{\mathcal{G}_t}$.
- (5) Add $B \rightarrow C$: $diff \gg 0$ because $C \rightarrow B \in \mathcal{G}_t$, so $\neg I(C, B|\emptyset)_{\mathcal{G}_t}$.
- (6) Add $B \rightarrow D$: $diff > 0$ 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 $C \rightarrow B$: $diff \gg 0$ because $C \rightarrow B \in \mathcal{G}_t$, so $\neg I(C, B|\emptyset)_{\mathcal{G}_t}$.
- (8) Add $C \rightarrow D$: $diff \gg 0$ 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 $D \rightarrow B$: $diff > 0$ because $\neg I(B, D|\emptyset)_{\mathcal{G}_t}$.
- (10) Add $D \rightarrow C$: $diff \gg 0$ because $D \rightarrow C \in \mathcal{G}_t$, so $\neg I(D, C|\emptyset)_{\mathcal{G}_t}$.

We can assume that FastCHC chooses to add $B \rightarrow C$ (Fig. 1b). The process continues, and those scores not affected by the last operation are retrieved from the cache. Thus, the only action to test is:

- (11) Add $D \rightarrow C$: $diff > 0$ because $\neg I(C, D|\{B\})_{\mathcal{G}_t}$.

At this point we can assume that FastCHC performs the addition operation with $C \rightarrow D$ (Fig. 1c). Again the scores not affected by the last operation are taken from the cache and not recomputed. The algorithm continues by analyzing the following action:

- (12) Add $B \rightarrow D$: $diff < 0$ because $I(B, D|\{C\})_{\mathcal{G}_t}$: $FP(B) = \{D\}$, $FP(D) = \{A, B\}$.

Addition $B \rightarrow A$ is now the operation selected at this step, but because of lines 15–20 in the FastCHC algorithm (Algorithm 3), FP sets for A and B must be updated besides the graph (Fig. 1d). Thus, now $FP(A) = \{D\}$, $FP(B) = \{D\}$ and $FP(C) = \emptyset$.

In the next step, FastCHC needs to study the operation:

- (13) Add $C \rightarrow A$: $diff > 0$ because $\neg I(A, C|\{B\})_{\mathcal{G}_t}$.

Then, FastCHC will include $A \rightarrow C$, yielding a minimal I-map (Fig. 1e). We should note that when the arc $B \rightarrow A$ was added, the sets $FP()$ were updated accordingly in order to delete the variable A from $FP(C)$, this operation being the only difference between the previous algorithms and FastCHC. Without updating those FP sets, $A \rightarrow C$ cannot be considered because $A \in FP(C)$.

The following proposition proves the type of output provided by the FastCHC algorithm.

Proposition 3 *Let D be a dataset containing d -independent and identically distributed samples from some distribution P . Let $\hat{G}$ be the directed acyclic graph obtained by running FastCHC algorithm by taking G_0 as the initial solution, i.e., $\hat{G} = \text{FastCHC}(G_0)$. If the score function f used to evaluate the candidate graphs in FastCHC is consistent and locally consistent, then under the assumption that P is faithful, $\hat{G}$ is a minimal I-map of P in the limit as d grows large.*

Proof in Appendix A.

5 Experimental evaluation

In this section we describe the set of experiments aimed at testing the performance of the algorithm presented in this paper. Given the nature of the proposed algorithm, we were not only interested in the quality of the solution obtained (accuracy) but also in the resources it needs (efficiency). Below we provide details about the implementation of the algorithms, the datasets used in this comparison, the performance indicators we have chosen to argue about the goodness of each algorithm, and to end the section we give the results and their analysis.

5.1 Algorithms

The algorithms examined in this section, apart from FastCHC, 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 starting graph. We do not consider the constrained hill climbing algorithm (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) [36] as the best scalable state-of-the-art structural learning algorithm for dealing with a high number of variables (though it does not guarantee an I-map is obtained).

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 furthermore the second stage of the algorithm is based on a taboo search rather than on canonical hill climbing, a custom implementation is used here. This implementation is based on the implementation of the MMPC algorithm available from [32], in language C, with the efficiency improvements described in [36]. 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) [25] with the same parameters as in [36], i.e. equivalent sample size equal to 10.

The implementation was coded in Java and interacts with the ProGraMo library for dataset and graph structures management [20].

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 table⁴ by using the names of the probability family as *key* and storing the score metric (real value) for

that family as *value*. Therefore, updating and querying operations over the cache are linear in time. To give the reader an idea of the benefits obtained by using this structure, the ratio $\frac{\text{\# of statistics required}}{\text{\# of actual calls to } f()}$ is 20, 25 and 75 for hill climbing algorithm when dealing with datasets/networks having 25, 30 and 100 variables (data from [18]).

All the runs of these algorithms were carried out on a dedicated server with Pentium Xeon 3.0 GHz, 64 bit architecture, 32 GB RAM memory and under Linux. The Java Runtime Environment used is the one provided by SUN Microsystems, version 1.5.

5.2 Networks

For this experimental comparison, we have selected the set of networks available in the Bayesian Network Repository (<http://www.cs.huji.ac.il/~galel/Repository/>). These are Alarm, Barley, HailFinder, Insurance, Link, Mildew, Munin (version 1, 2, 3 and 4), Pigs, and PathFinder (version 1 and 2.3). The Diabetes network is not used because of its dynamic nature and Carpo because it contains non-standard probability functions. More detailed information about these networks is given in Table 1.

For all these networks, we obtained different datasets by sampling with 500, 1,000 and 5,000 instances. Each dataset was given the same name as its corresponding network. In fact, six datasets are sampled for each network and size: one is used for learning and the other five for validation (the average is reported).

5.3 Performance indicators

Two kinds of factors are used as performance indicators, one being the quality of the network obtained by the algorithm, and the other the complexity of each algorithm.

The first group contains two different statistics that, respectively, compare the learned network regarding a dataset and the learned network regarding the golden model, i.e., the network used to generate the datasets:

- *Loglikelihood* Given a dataset $D = \{\mathbf{v}^1, \dots, \mathbf{v}^m\}$ of instances and a network $\mathcal{G}$ defined over a set of variables $V = \{X_1, \dots, X_n\}$, the log-likelihood of $\mathcal{G}$ with respect to D is computed as:

$$LL(\mathcal{G} : D) = \sum_{i=1}^m \log(P_{\mathcal{G}}(\mathbf{v}_i | D)).$$

- *Structural Hamming Distance (SHD)* Given two models $\mathcal{G}_1$ and $\mathcal{G}_2$ (the golden one and the learned one in our case) the SHD is computed over their essential graphs $eg(B_1)$

⁴ Concretely, java language `HashMap` class is used (<http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html>).

Table 1 Bayesian networks used in our experimental evaluation

	Description	Nodes	Edges	States per node	Max states	Parents per node	Min–max parents	PC per node	Max PC
Alarm	Monitoring of emergency care patients	37	46	2.84	2–4	1.24	4	2.49	6
Barley	Model of barley crop yields	48	84	8.77	2–67	1.75	4	3.50	8
Diabetes	A model for insulin dose adjustment (DBN)	413	602	11.34	3–21	1.46	2	2.92	24
Hailfinder	Predicting hail in northern Colorado	56	66	3.98	2–11	1.18	4	2.36	17
Insurance	Evaluating insurance applications	27	52	3.30	2–5	1.93	3	3.85	9
Link	Pedigree for linkage analysis	724	1,125	2.53	2–4	1.55	3	3.11	17
Mildew	A model for deciding on the amount of fungicides to be used against attack of mildew in wheat	35	46	17.6	3–100	1.31	3	2.63	5
Munin1	An expert electromyography assistant	189	282	5.26	1–21	1.49	3	2.98	15
Munin2		1,003	1,244	5.36	2–21	1.24	3	2.48	30
Munin3		1,044	1,315	5.37	1–21	1.26	3	2.52	69
Munin4		1,041	1,397	5.43	1–21	1.34	3	2.68	69
Pf1	Analysis of lymph cell pathologies	109	195	4.11	2–63	1.79	5	3.58	106
Pf23		135	200	3.85	2–76	1.48	4	2.96	130
Pigs	Pedigree of breeding pigs	441	592	3.00	3–3	1.34	2	2.68	41
Water	A model of the biological processes of a water purification plant	32	66	3.63	3–4	2.06	5	4.13	8
Win95pts	A model for printer troubleshooting in Microsoft Windows 95	76	112	2.00	2–2	1.47	7	2.95	10

For each network we indicate a description, the number of variables, edges, average number of states, minimum and maximum number of states, average number of parents per variable, maximum number of parents, average number of parents and children (PC) per variable and maximum number of parents and children

and $eg(B_2)$ in order to not penalize structural differences that cannot be statistically distinguished. The algorithm described in [6] is used to transform a given DAG into an essential graph or *partially oriented graph* (PDAG). Then,

$$SHD(eg(\mathcal{G}_1) : eg(\mathcal{G}_2)) = |eg(\mathcal{G}_2) - eg(\mathcal{G}_1)| + |eg(\mathcal{G}_1) - eg(\mathcal{G}_2)| + |diff(\mathcal{G}_1, \mathcal{G}_2)|$$

that is, SHD is computed as the number of edges missing in the first graph with respect to the second, plus the number of extra edges in the first graph with respect to 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 detected in the other. See [36] for details.

With regard to the efficiency factor, the number of computations carried out by each algorithm, i.e. statistical tests or score function calls, is collected, as this has a direct correspondence with the CPU time required. It is important to

notice that given the differences in programming language, i.e. C++ and Java, the run time is not a fair indicator.

5.4 Results

In the following tables, we present a summary of the results we obtained after running the 6 algorithms over the 16 domains (see Table 1). The summarand also the Structural Hamming Distance (SHD) with respect to the golden model (the one used to sample training and test sets). SHD is measured as in [36], based on comparing the essential graphs.

The structural Hamming distance is computed for a pair of partially directed acyclic graphs (PDAGs). Thus, the output models obtained by each algorithm are converted into a PDAG. Then, for every pair of graphs, SHD is computed as the number of edges missing in the first graph with respect to the second, plus the number of extra edges in the first graph with respect to 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. y consists of the average rank of each algorithm in each dataset, from 1 to 6, 1 being the best result. The complete list of results

Table 2 Average rank of the algorithms based on likelihood over all the validation datasets

Algorithm	500	1,000	5,000
CHC*	1.94 ^a	1.84 ^a	2.16 ^a
FastCHC	4.63	3.78	3.60
HC	1.13	1.31	1.72
iCHC	3.84	4.16	3.94
MMHC	5.38	5.50	5.47
2iCHC	4.09	4.41	4.13

The best results are highlighted in bold

^a Those results not significantly worse than the best one

can be seen in Appendix B. The MMHC algorithm could not handle many of the big datasets, especially those with 5,000 instances. The reason is that the algorithm needs more memory than that available in the computer, or the process does not finish before the set limit of 1 week.⁵ In these cases, we consider that the algorithm gives a worse result than the others in order to penalize the fact that it cannot produce a result under the given conditions.

For each one of the indicators, we performed a Friedman test [15] in order to discover whether we can assume that all algorithms deliver the same performance, and in the case that we cannot assume so we use the Holm's post hoc test [26] to see which algorithms can be taken as not statistically worse than the best one. In all the tests, the significance level is set to 0.05.

In Table 2, we can see the results for the averaged likelihood of the obtained network for the five dataset sampled for validation. The Friedman test rejects the null hypothesis in all cases. Regardless of the size of the dataset, the best algorithm is always hill climbing, with CHC* not being significantly worse. MMHC is the worst in all cases. It is important to notice that FastCHC improves its rank as the dataset increases, from 4.63 to 3.60, being the only algorithm that has this improvement consistently as the size of dataset increases.

Table 3 shows the second quality factor, SHD. Here the Friedman test rejects the null hypothesis in all cases as well. As in the previous case, hill climbing is the best algorithm, but now not only CHC* produces comparable results. FastCHC is also comparable to hill climbing in all sizes and iCHC and MMHC in the smaller ones.

Looking at these two result tables together we can draw the conclusion that the HC algorithm probably obtains such good results in likelihood because it overfits the network model to the data by adding many additional arcs that in turn deteriorate its performance in terms of SHD.

⁵ Notice that we do not set a maximum number of parents during the search.

Table 3 Average rank of the algorithms based on SHD over all the datasets

Algorithm	500	1,000	5,000
CHC*	3.19 ^a	2.91 ^a	2.78 ^a
FastCHC	3.31 ^a	3.31 ^a	3.09 ^a
HC	2.38	2.50	2.09
iCHC	4.53	4.50	4.28
MMHC	3.44 ^a	3.41 ^a	4.66
2iCHC	4.16	4.38	4.09

The best results are highlighted in bold

^a Those results not significantly worse than the best one

Table 4 Average rank of the algorithms based on the statistics computed over all the datasets

Algorithm	500	1,000	5,000
CHC*	4.63	4.44	4.19
FastCHC	1.19	1.19	1.00
HC	5.75	5.69	5.31
iCHC	3.16	3.09	2.78
MMHC	3.94	4.19	5.50
2iCHC	2.34 ^a	2.41 ^a	2.22 ^a

The best results are highlighted in bold

^a Those results not significantly worse than the best one

Table 5 Average rank of the algorithms based on the combined score of likelihood and statistics computed over all the datasets

Algorithm	500	1,000	5,000
CHC*	4.44	4.31	4.19
FastCHC	1.19	1.06	1.00
HC	5.69	5.63	5.25
iCHC	3.16	3.09	2.78
MMHC	4.19	4.46	5.56
2iCHC	2.34 ^a	2.34 ^a	2.22 ^a

The best results are highlighted in bold

^a Those results not significantly worse than the best one

Finally, we study the efficiency indicator, namely the number of statistics (scores) computed, as this number is proportional to CPU time. In Table 4, we can see that the most efficient algorithm is FastCHC. The hill climbing algorithm is the least efficient in general. In this case, the Friedman test suggests that the null hypothesis has to be rejected, and the post hoc test indicates that only 2iCHC is not significantly worse than FastCHC.

In summary, the above tables seem to indicate that the FastCHC algorithm is the best in terms of scalability and efficiency, while HC is the best algorithm in terms of the quality of the model.

However, even though the output obtained by HC is sometimes better than that obtained by FastCHC, the cost

associated with this improvement may not be worth it. To support this statement, in Table 5 we show the combined result of the resulting model's likelihood and the effort required to obtain it. This is 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. It can be formulated 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 statistics computed by the algorithm i on the dataset j . This factor indicates how much benefit per computation is produced by each algorithm.

To end with the analysis of the results and as a sort of graphical summary, we represent in Fig. 2a comparison of the results obtained by the six algorithms over three dimensions: (log)likelihood, structural Hamming distance (SHD) and number of computed statistics. To allow for a fast and visual comparison, we average the result over all the datasets for each algorithm, and take the inverse when necessary, in such a way, that in all the cases, the greater the value, the worse the behaviour of the algorithm. As we can observe from Fig. 2 (and also from full results shown in Appendix B), the differences in two of the three axes (loglikelihood and SHD) are small, while the big difference appears in the third axe (number of statistics), what graphically reinforces the analysis exposed above.

6 Conclusions

In this paper, we have proposed an improved version of the algorithms in the CHC family, which in turn are improved versions of the standard hill climbing approach for learning Bayesian networks. This new algorithm avoids the necessity of iterating the search algorithm by updating the constrained search space after applying the local action/operation selected at each step. This behaviour prevents the possibility that the marginally independent variables will be considered as conditionally independent when in the actual (underlying) probability distribution there are not.

Obviously, as the FastCHC algorithm requires only one iteration, it is much more efficient than previous CHC algorithms. This is clear in our experimental evaluation, in which the superiority of FastCHC over MMHC is also made evident.

Therefore, FastCHC constitutes a good candidate for learning BNs in high dimensional problems, in which scalable algorithms are required. Furthermore, because the speed up achieved with respect to competing algorithms we think that there is room to improve the quality of the discovered network, by slightly sacrificing the speed of the search. That is, for the future we plan to search more in order to search

Fig. 2 Graphical comparison of the tested algorithms over three dimensions: (log) likelihood, structural Hamming distance and running time (#statistics). Three different sample sizes are studied (500, 1,000 and 5,000)

better. Thus, the idea is to use FastCHC as the basic building block for using some local search-based meta-heuristics like taboo search or simulated annealing, which have proven to be able to escape from local optima.

Acknowledgments This work has been partially funded by FEDER funds and the Spanish Government (MICINN/MINECO) through project TIN2010-20900-C04-03.

Appendix A: Proofs for Proposition 3

In this appendix we prove Proposition 3, but first we need the following results. In all cases, we consider the following conditions:

Let D be a dataset containing m iid samples from some distribution p . Let $\hat{\mathcal{G}}$ be the DAG obtained by running the learning algorithm and taking a DAG $\mathcal{G}_0$ as the initial solution. Let us consider a consistent and locally consistent metric f to evaluate DAGs during the search, and finally we consider the number of cases m grows large.

Proposition 4 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 $\mathcal{G}$. This proposition is proven in [17].

Proposition 5 Under the conditions stated above, with $\mathcal{G}^*$ being the graph that the distribution p was sampled from, FastCHC (Algorithm 3) 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 FastCHC because there is no subset $\mathbf{S} \subset \mathbf{V}$ such that $I_p(X_i, X_j | \mathbf{S})$. Specifically we have $\neg I_p(X_i, X_j | \mathbf{Pa}_{\hat{\mathcal{G}}}(X_i))$, given any subset of parents $\mathbf{Pa}_{\hat{\mathcal{G}}}(X_i)$, and so $f_D(X_{p_i}, \mathbf{Pa}_{\hat{\mathcal{G}}}(X_{p_i}) \cup \{X_j\}) - f_D(X_{p_i}, \mathbf{Pa}_{\hat{\mathcal{G}}}(X_{p_i})) > 0$ for any subset $\mathbf{Pa}_{\hat{\mathcal{G}}}(X_i)$. As the reverse is also true, that is, $f_D(X_{p_j}, \mathbf{Pa}_{\hat{\mathcal{G}}}(X_{p_j}) \cup \{X_i\}) - f_D(X_{p_j}, \mathbf{Pa}_{\hat{\mathcal{G}}}(X_{p_j})) > 0$, we can conclude that, at some stage in 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 FastCHC 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 actual graph, so the graph $\hat{\mathcal{G}}$

by the algorithm will include the v-structure. In the case of one edge being wrongly 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 leads to the inclusion of edge $X_a \rightarrow X_b$, i.e., 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 in lines 15–20 of the algorithm because X_a will be removed from $FP(X_b)$ and 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). $\square$

Proposition 6 Under the conditions stated above, with $\mathcal{G}^*$ being the graph from which the distribution p was sampled, FastCHC (Algorithm 3) 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, i.e., 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 the fact that:

- 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 $\hat{\mathcal{G}}$, thus contradicting 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 $\neg(X_a, X_b | \mathbf{S})_{\mathcal{G}^*}$, for a 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 in lines 15 to 20 of the algorithm because X_a will be removed from $FP(X_b)$ and never be included in $FP(X_b)$, and viceversa, because of X_c opens the path between X_a and X_b as long as it remains as parent of X_b , and viceversa (note that if edge $X_c \rightarrow X_b$ is reversed then we obtain the v-structure). $\square$

Proposition 7 Let D be a dataset containing d -independent and identically distributed samples from some distribution P . Let $\hat{\mathcal{G}}$ be the directed acyclic graph obtained by running the FastCHC algorithm and taking $\mathcal{G}_0$ as the initial solution, i.e., $\hat{\mathcal{G}} = \text{FastCHC}(\mathcal{G}_0)$. If the score function f used to evaluate the candidate graphs in FastCHC is consistent and locally consistent, then under the assumption that P is faithful, $\hat{\mathcal{G}}$ is a minimal I-map of P in the limit as d grows large.

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 FastCHC algorithm and $\mathcal{G}^*$ the graph from which distribution p is obtained. The second step is to prove that $\hat{\mathcal{G}}$ is in fact a *minimal* I-map of $\mathcal{G}^*$. [(1) $\hat{\mathcal{G}}$ is an I-map of $\mathcal{G}^*$] This is straightforward from Propositions 4, 5 and 6.

[(2) minimal condition] Let us suppose the contrary, i.e., there exists $X_j \in \mathbf{Pa}_{\hat{\mathcal{G}}}(X_i)$ such that $I_p(X_i, X_j | \mathbf{Pa}_{\hat{\mathcal{G}}}(X_i) \setminus \{X_j\})$.

If so, $\hat{\mathcal{G}}$ cannot be a local optimum because there is (at least) one deletion operation with a positive difference. $\square$

Appendix B: Extended results

See Tables 6, 7, 8, 9, 10, 11, 12, 13 and 14.

Table 6 Averaged over the five test dataset, the log-likelihood obtained for each model learned with each algorithm with the dataset with 500 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	−7,616	−7,639	−7,538	−7,629	−7,779	−7,629
Diabetes	−138,863	−183,942	−131,228	−182,071	NA	−182,071
Barley	−40,612	−42,193	−40,612	−41,719	−41,105	−41,719
Hailfinder	−25,495	−25,757	−25,495	−25,757	−26,408	−25,757
Insurance	−9,833	−10,222	−9,779	−10,222	−10,026	−10,222
Link	−114,358	−123,591	−113,998	−125,632	−126,305	−125,661
Mildew	−22,322	−22,641	−22,322	−22,641	−24,194	−22,641
Munin1	−35,006	−35,635	−34,971	−35,575	−41,336	−35,575
Munin2	−153,754	−155,309	−153,618	−155,281	NA	−155,281
Munin3	−150,995	−152,781	−150,788	−152,545	NA	−152,545
Munin4	−156,637	−158,243	−156,299	−158,240	NA	−158,244
Pf1	−42,241	−42,354	−42,230	−42,290	−44,988	−42,290
Pf23	−54,851	−54,927	−54,847	−54,871	−60,126	−54,871
Pigs	−172,078	−173,445	−171,866	−175,514	−171,887	−175,514
Water	−9,838	−9,886	−9,838	−9,886	−9,921	−9,886
Win95pts	−8,778	−9,033	−8,771	−8,934	−10,408	−8,999

Table 7 Averaged over the five test dataset, the log-likelihood obtained for each model learned with each algorithm with the dataset with 1,000 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	−15,047	−15,106	−15,046	−15,142	−15,514	−15,144
Barley	−78,971	−81,255	−78,971	−81,255	−81,490	−81,255
Diabetes	−278,124	−359,285	−260,346	−358,935	NA	−358,935
Hailfinder	−51,416	−52,186	−51,416	−52,186	−53,446	−52,186
Insurance	−19,871	−20,433	−19,658	−20,466	−20,370	−20,466
Link	−230,359	−243,715	−229,576	−249,213	−246,520	−249,238
Mildew	−43,441	−43,782	−43,441	−43,782	−47,184	−43,782
munin1	−71,376	−72,227	−71,122	−72,242	−83,312	−72,242
Munin2	−310,733	−314,480	−310,572	−313,659	NA	−313,659
Munin3	−305,290	−307,569	−305,143	−308,485	NA	−308,485
Munin4	−314,716	−317,131	−314,531	−316,946	NA	−316,946
Pf1	−84,865	−85,396	−84,865	−85,414	−88,472	−85,424
Pf23	−108,848	−108,915	−108,877	−108,998	−122,447	−108,998
Pigs	−345,805	−345,794	−345,794	−345,805	−345,806	−345,805
Water	−19,738	−19,818	−19,738	−19,818	−19,796	−19,818
Win95pts	−17,873	−18,351	−17,890	−18,174	−20,552	−18,237

Table 8 Averaged over the 5 test dataset, the log-likelihood obtained for each model learned with each algorithm with the dataset with 5,000 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	−77,653	−77,748	−77,653	−77,748	−78,020	−77,748
Barley	−373,425	−391,241	−373,425	−391,241	NA	−391,241
Diabetes	NA	−1501,579	NA	−1,494,135	NA	−1,507,244
Hailfinder	−257,288	−257,523	−257,288	−257,831	−296,571	−257,831
Insurance	−98,465	−100,652	−98,465	−100,861	−98,917	−100,861
Link	−1,152,755	−1,189,608	−1,152,827	−1,222,721	−1,196,949	−1,222,721
Mildew	−202,444	−215,757	−201,347	−215,757	NA	−215,757
Munin1	−359,560	−360,363	−359,124	−360,253	NA	−360,253
Munin2	−1,556,601	−1,558,260	−1,555,881	−1,559,764	NA	−1,559,764
Munin3	−1,539,425	−1,541,086	−1,538,133	−1,542,840	NA	−1,542,840
Munin4	−1,587,786	−1,590,419	−1,586,574	−1,591,233	NA	−1,591,233
Pf1	−399,203	−402,629	−394,994	−404,791	NA	−404,791
Pf23	−522,024	−522,141	−521,971	−522,055	NA	−522,055
Pigs	−1,734,761	−1,734,761	−1,734,761	−1,734,761	−1,734,761	−1,734,761
Water	−98,737	−98,737	−98,737	−98,737	−98,784	−98,737
Win95pts	−91,653	−93,410	−91,646	−92,519	−102,699	−92,867

Table 9 Structural Hamming distance of the network learned by each algorithm with 500 instances compared to the original model

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	42	37	39	38	16	38
Barley	73	79	73	78	74	78
Diabetes	393	447	304	459	NA	459
Hailfinder	26	27	26	28	32	28
Insurance	27	33	25	33	26	33
Link	1,029	1,113	971	1,174	746	1,172
Mildew	34	36	34	36	36	36
Munin1	240	235	229	238	237	237
Munin2	1,112	1,117	1,101	1,122	NA	1,122
Munin3	1,125	1,137	1,103	1,141	NA	1,141
Munin4	1,167	1,169	1,137	1,185	NA	1,184
Pf1	221	209	228	215	185	214
Pf23	258	250	257	255	217	255
Pigs	42	56	36	103	7	103
Water	45	49	45	49	53	49
Win95pts	224	213	228	220	90	213

Table 10 Structural Hamming distance of the network learned by each algorithm with 1,000 instances compared to the original model

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	36	27	37	30	14	29
Barley	70	75	70	75	73	75
Diabetes	342	392	276	397	NA	397
Hailfinder	25	32	25	32	37	32
Insurance	27	31	22	33	24	33
Link	701	796	652	854	600	855
Mildew	24	25	24	25	26	25
Munin1	209	204	185	211	208	211
Munin2	881	902	883	909	NA	909
Munin3	835	843	805	851	NA	851
Munin4	1,008	1,035	985	1,063	NA	1,063
Pf1	206	197	206	209	195	208
Pf23	224	220	223	221	208	221
Pigs	3	11	11	3	0	3
Water	40	44	40	44	44	44
Win95pts	210	192	209	198	72	195

Table 11 Structural Hamming distance of the network learned by each algorithm with 5,000 instances compared to the original model

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	4	5	4	5	4	5
Barley	42	55	42	55	NA	55
Diabetes	NA	315	NA	340	NA	318
Hailfinder	20	25	20	25	43	25
Insurance	15	17	15	19	15	19
Link	569	632	563	816	488	816
Mildew	19	22	13	22	NA	22
Munin1	133	137	129	145	NA	145
Munin2	767	794	758	799	NA	799
Munin3	694	708	667	721	NA	721
Munin4	804	825	772	832	NA	832
Pf1	175	168	162	177	NA	177
Pf23	163	159	162	161	NA	161
Pigs	0	0	0	0	0	0
Water	32	32	32	32	33	32
Win95pts	108	101	102	110	74	106

Table 12 Number of computations made to learn the network by each algorithm with 500 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	2,519	1,410	3,953	1,836	2,380	1,782
Barley	2,894	1,276	3,313	1,707	2,497	1,699
Diabetes	290,771	93,934	428,819	166,120	NA	161,751
Hailfinder	4,311	2,004	6,247	2,560	4,523	2,526
Insurance	1,218	597	1,737	788	1,382	788
Link	1,017,315	627,929	1,339,409	735,811	493,099	730,128
Mildew	1,613	713	2,078	986	1,204	986
Munin1	67,737	41,302	91,854	53,124	29,367	51,387
Munin2	1,676,174	1,155,527	2,138,695	1,405,095	NA	1,401,990
Munin3	1,855,462	1,271,087	2,364,820	1,545,354	NA	1,535,154
Munin4	1,824,202	1,228,419	2,311,309	1,505,846	NA	1,498,908
Pf1	19,376	7,730	29,146	12,209	18,722	11,852
Pf23	26,545	10,609	37,079	17,171	25,292	16,794
Pigs	280,625	105,793	472,044	157,124	248,222	156,292
Water	1,522	980	1,858	1,182	992	1,182
Win95pts	16,380	9,135	24,315	11,010	6,503	10,032

Table 13 Number of computations made to learn the network by each algorithm with 1,000 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	2,804	1,376	3,886	1,836	2,520	1,756
Barley	2,987	1,317	3,587	1,799	2,678	1,799
Diabetes	285,347	96,725	422,264	169,090	NA	167,546
Hailfinder	4,390	2,044	6,194	2,556	4,514	2,556
Insurance	1,288	644	1,857	840	1,638	840
Link	936,673	556,195	1,184,389	655,432	527,127	651,553
Mildew	1,587	735	2,239	996	1,529	996
Munin1	64,255	39,671	87,577	50,921	34,806	49,945
Munin2	1,581,454	1,074,713	1,974,659	1,310,124	NA	1,305,591
Munin3	1,770,587	1,217,225	2,189,912	1,459,869	NA	1,454,323
Munin4	1,758,639	1,186,447	2,188,804	1,451,041	NA	1,438,270
Pf1	19,698	7,945	28,181	11,778	25,584	11,242
Pf23	24,785	10,949	35,082	15,488	35,384	15,338
Pigs	245,203	107,109	463,264	149,490	399,768	149,141
Water	1,498	1,009	1,885	1,167	1,144	1,167
Win95pts	14,638	7,660	22,786	9,725	6,961	9,101

Table 14 Number of computations made to learn the network by each algorithm with 5,000 instances

	CHC*	FastCHC	HC	iCHC	MMHC	2iCHC
Alarm	2,048	1,210	3,027	1,491	2,749	1,491
Barley	3,367	1,337	4,494	1,880	NA	1,880
Diabetes	NA	111,855	NA	194,607	NA	176,871
Hailfinder	4,426	2,242	6,567	2,724	4,589	2,724
Insurance	1,351	759	1,882	928	2,835	928
Link	920,866	498,798	1,125,956	594,659	639,839	588,567
Mildew	1,745	814	2,497	1,099	NA	1,099
Munin1	60,972	37,924	82,548	48,309	NA	47,941
Munin2	1,513,360	1,027,470	1,868,651	1,255,838	NA	1255176
Munin3	1,713,468	1, 219,925	2,103,503	1,448,472	NA	1,447,329
Munin4	1,700,898	1,190,055	2,135,861	1,427,648	NA	1,426 ,626
Pf1	22,410	10,179	29,033	13,553	NA	13,321
Pf23	26,054	13,484	35,876	17,469	NA	17,349
Pigs	249,920	114,072	454,073	155,141	11,189,385	155,141
Water	1,562	1,154	2,059	1,300	1,727	1,300
Win95pts	10,227	5,229	15,395	6,586	7,506	6,421

References

1. Acid, S., de Campos, L.M.: A hybrid methodology for learning belief networks: Benedict. *Int. J. Approx. Reason.* **27**(3), 235–262 (2001)
2. Acid, S., de Campos, L.M.: Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs. *J. Artif. Intell. Res.* **18**, 445–490 (2003)
3. Buntine, W.L.: Theory refinement on Bayesian networks. In: *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 52–60 (1991)
4. Buntine, W.L.: A guide to the literature on learning probabilistic networks from data. *IEEE Trans. Knowl. Data Eng.* **8**(2), 195–210 (1996)
5. Cano, R., Sordo, C., Gutiérrez, J.M.: Applications of Bayesian networks in meteorology. In: Gámez, J.A., Moral, S., Salmerón, A. (eds.) *Advances in Bayesian Networks*, pp. 309–327. Springer, Berlin (2004)
6. Chickering, D.M.: A transformational characterization of equivalent Bayesian network structures. In: *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 87–98. Morgan Kaufmann, San Francisco (1995)
7. Chickering, D.M.: Learning Bayesian networks is NP-Complete. In: Fisher, D., Lenz, H. (eds.) *Learning from Data: Artificial Intelligence and Statistics*, vol. V, pp. 121–130. Springer, Berlin (1996)
8. Chickering, D.M., Geiger, D., Heckerman, D.: Learning Bayesian networks: search methods and experimental results. In: *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pp. 112–128 (1995)
9. Chickering, D.M.: Optimal structure identification with greedy search. *J. Mach. Learn. Res.* **3**, 507–554 (2002)
10. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. *Mach. Learn.* **9**, 309–347 (1992)
11. de Campos, L.M.: A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *J. Mach. Learn. Res.* **7**, 2149–2187 (2006)
12. de Campos, L.M., Fernández-Luna, J.M., Gámez, J.A., Puerta, J.M.: Ant colony optimization for learning Bayesian networks. *Int. J. Approx. Reason.* **31**(3), 291–311 (2002)
13. de Campos, L.M., Fernández-Luna, J.M., Puerta, J.M.: Local search methods for learning Bayesian networks using a modified neighborhood in the space of dags. In: *Proceedings of IBERAMIA 2002. LNCS*, vol. 2527, pp. 182–192 (2002)
14. de Campos, L.M., Puerta, J.M.: 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 (2001)
15. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.* **32**(200), 675701 (1937)
16. Friedman, N., Nachman, I., Pe'er, D.: 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 (1999)
17. Gámez, J.A., Mateo, J.L., Puerta, J.M.: Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining Knowl. Discov.* **22**(1–2), 106–148 (2011)
18. Gámez, J.A., Puerta, J.M.: Constrained score+(local)search methods for learning Bayesian networks. In: *8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'05)*, pp. 161–173 (2005)
19. Gámez, J.A., Puerta, J.M.: 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 (2005)
20. Gámez, J.A., Salmerón, A., Cano, A.: Design of new algorithms for probabilistic graphical models. implementation in elvira. *programo research project (tin2007-67418-c03)*. In: *Jornada de Seguimiento de Proyectos*, 2010. Programa Nacional de Tecnologías Informáticas (2010)

21. García-Pedrajas, N., de Haro-García, A.: Scaling up data mining algorithms: review and taxonomy. *Prog. Artif. Intell.* **1**, 71–87 (2012)
22. Geiger, D., Heckerman, D., King, H., Meek, C.: Stratified exponential families: graphical models and model selection. *Ann. Stat.* **29**(2), 505–529 (2001)
23. Haughton, D.M.A.: On the choice of a model to fit data from an exponential family. *Ann. Stat.* **16**(1), 342–355 (1988)
24. Heckerman, D.: Bayesian networks for data mining. *Data Mining Knowl. Discov.* **1**, 79–119 (1997)
25. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. *Mach. Learn.* **20**(3), 197–243 (1995)
26. Holm, S.: A simple sequentially rejective multiple test procedure. *Scand. J. Stat.* **6**, 65–70 (1979)
27. Jensen, F.V., Nielsen, T.D.: *Bayesian Networks and Decision Graphs*, 2nd edn. Springer, Berlin (2007)
28. Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R.H., Kuijpers, C.M.H.: Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters. *IEEE Trans. Pattern Anal. Mach. Intell.* **18**(9), 912–926 (1996)
29. Nägele, A., Dejori, M., Stetter, M.: 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 (2007)
30. Neapolitan, R.: *Learning Bayesian Networks*. Prentice Hall, New Jersey (2003)
31. Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco (1988)
32. Peña, J.M., Nilsson, R., Björkegren, J., Tegnér, J.: Towards scalable and data efficient learning of Markov boundaries. *Int. J. Approx. Reason.* **45**(2), 211–232 (2006)
33. Robinson, R.W.: Counting unlabeled acyclic digraphs. In: *Combinatorial Mathematics*, vol. 622, pp. 28–43. Springer, Berlin (1977)
34. Schwarz, G.: Estimating the dimension of a model. *Ann. Stat.* **6**(2), 461–464 (1978)
35. Spirtes, P., Glymour, C., Scheines, R.: Causation, prediction and search. In: *Lecture Notes in Statistics*, vol. 81. Springer, Berlin (1993)
36. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max–min hill-climbing Bayesian network structure learning algorithm. *Mach. Learn.* **65**(1), 31–78 (2006)
37. van Dijk, S., van der Gaag, L.C., Thierens, D.: A skeleton-based approach to learning Bayesian networks from data. In: *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, pp. 132–143 (2003)
38. Verma, T., Pearl, J.: Equivalence and synthesis of causal models. In: *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI'90)*, pp. 255–270. Elsevier, Amsterdam (1991)
39. Wong, M.L., Leung, K.-S.: An efficient data mining method for learning Bayesian networks using an evolutionary algorithm-based hybrid approach. *IEEE Trans. Evol. Comput.* **8**(4), 378–404 (2004)