International Journal of Approximate Reasoning 54 (2013) 429–451

Contents lists available at SciVerse ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

Scaling up the Greedy Equivalence Search algorithm by constraining the search space of equivalence classes

Juan I. Alonso-Barba *, Luis delaOssa, Jose A. Gámez, Jose M. Puerta

Department of Computing Systems, Intelligent Systems and Data Mining Lab, Albacete Research Institute of Informatics, University of Castilla-La Mancha, 02071 Albacete, Spain

ARTICLE INFO

Article history:
Available online 13 October 2012

Keywords:
Bayesian networks
Greedy Equivalence Search
Constrained search

ABSTRACT

Greedy Equivalence Search (GES) is nowadays the state of the art algorithm for learning Bayesian networks (BNs) from complete data. However, from a practical point of view, this algorithm may not be efficient enough to deal with data from high dimensionality and/or complex domains. This paper proposes some modifications to GES aimed at increasing its efficiency. Under the faithfulness assumption, the modified algorithms preserve the same theoretical properties of the original one, that is, they recover a perfect map of the target distribution in the large sample limit. Moreover, experimental results confirm that, although the proposed methods carry out a significantly smaller number of computations, the quality of the BNs learned can be compared with those obtained with GES.

1. Introduction

Bayesian network (BN) learning [1] has received growing attention from researchers in the field of data mining. This interest is twofold: first, after obtaining the BN model from data, it can be used for descriptive representation of dependencies among domain variables; secondly, the obtained models can also be used for prediction, since there are efficient inference mechanisms specifically developed to carry out this task on BNs.

Although many approaches and methods have been proposed so far, there is still great interest in developing new algorithms. This is due to the fact that BN learning is a complex and highly time-consuming task, and most of the existing methods usually do not scale up when dealing with complex domains where a huge number of variables are considered. Therefore, new algorithms, as well as improvements to the existing ones, keep on being proposed so as to work with bigger datasets [2].

There are two main approaches for structural learning of BNs:

- *Score + search methods*. These kinds of algorithms use a function f to measure the goodness of a network with respect to the training data, and a search method to find the network with the best score.

  Learning the structure of a Bayesian network is an NP-Hard problem [3]. Thus, although exact techniques [4] can be used to carry out this task, these algorithms can only deal with a very limited number of variables (the record is currently 26). Because of that, the search is usually carried out by means of heuristic methods. Due to their efficiency and effectiveness, those based on local search [5–8,1, Chapter 9] are the most commonly used, particularly when dealing with complex domains, since there are improvements [7,9] which make it possible to avoid a huge number of computations. Nevertheless, other metaheuristics, such as genetic algorithms [10], simulated annealing [8], tabu search [11], ant colony optimization [12], and estimation of distribution algorithms [13] have also been proposed.

* Corresponding author.
E-mail addresses: JuanIgnacio.Alonso@uclm.es (J.I. Alonso-Barba), Luis.delaOssa@uclm.es (L. delaOssa), Jose.Gamez@uclm.es (J.A. Gámez), Jose.Puerta@uclm.es (J.M. Puerta).

0888-613X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ijar.2012.09.004

In relation to the scoring metrics, the ones most commonly used in the literature are: the Bayesian Dirichlet equivalent (BDe) [5], the Minimum Description Length (MDL) [14] and the Bayesian Information Criterion (BIC) [15]; however, more alternatives exist, such as MIT [16].

- *Constraint-based methods*. The idea underlying these methods is to satisfy as much independence present in the data as possible [17,1, Chapter 10]. Statistical hypothesis testing is used to determine the validity of conditional independence sentences.

Finally, there also exist hybrid algorithms that combine these previous two approaches [18].

In general, the aforementioned methods are defined over the space of Directed Acyclic Graphs (DAGs) or *D-space*. Despite being very efficient, these algorithms do not achieve the results, in terms of network score, obtained by those methods defined over the space of Equivalence Classes (*E-space*) [5,6]. The most representative of such methods, GES (Greedy Equivalence Search) [6], is nowadays considered as the reference algorithm in BN learning since, under faithfulness conditions, it asymptotically obtains a perfect map of the target distribution. Some variants of GES have also been proposed, such as the KES algorithm [19] or a previous work by the current authors in which some of the algorithms presented in this article were firstly proposed [20].

As its name indicates, GES is also a local (greedy) algorithm. In a first stage, GES takes the equivalence class with no dependences as a starting point, and adds edges until all dependences are gathered in the model; then, it tries to progressively delete edges in a second stage. Despite considering only local changes in both phases, the algorithm has several drawbacks which limit its scalability. Thus, the cost of the algorithm is exponential in the size of the biggest clique in the network, since it is necessary to carry out an exhaustive search to find the best subset of head-to-head nodes with the tested edges in both phases. Therefore, GES is very inefficient when dealing with structures where a node has a large number of adjacent nodes.

Although the theoretical complexity of both phases is exponential, it is the first stage that limits the applicability of the algorithm from a practical point of view, since the second one usually runs considerably fewer iterations, and the size of the cliques found in the second stage is smaller. Therefore, it is common to restrict the maximum number of parents of a node. However, this decision is arbitrary, and an incorrect setting of this parameter may degrade the performance of the algorithm.

In this work, we propose several alternatives for improving the efficiency of GES. Since most computational effort is made during the constructive stage, it would be of interest to relax the search of the head-to-head subsets for each node. In order to do that, we propose changing the exhaustive search, whose complexity is exponential in the number of adjacencies for a node, for a greedy one, which is linear. In this line, several proposals have been studied and tested.

Furthermore, it is also possible to improve the scalability of GES by methods successfully used in the *D-space* [7,9,21], which mainly consist of dynamically pruning the search space. Thus, computations carried out at each search step of the constructive phase allow the detection of edges which should not be considered from then on.

As we show in this paper, the modifications to GES incorporating both changes exhibit the same theoretical properties as the original algorithm. Moreover, experiments confirm that they speed up the unconstrained/classical version, and some of them are able to learn BNs of the same quality.

This paper is structured as follows. Section 2 provides an introduction to the main concepts which are covered in the paper. In Sections 3–5 we introduce the BN learning problems; the main properties that a metric scoring function requires in order to prove the correctness of the algorithms studied in the paper; and a detailed description of GES. Our proposals for scaling up GES are then presented in Section 6, and an experimental analysis is carried out in Section 7. Lastly, Section 8 presents some conclusions.

2. Preliminaries

Bayesian networks are probabilistic graphical models that efficiently represent and manipulate n-dimensional probability distributions [22]. A BN can be defined as a pair $\mathcal{B} = (\mathcal{G}, \Theta)$ such that:

- $\mathcal{G} = (\boldsymbol{V}, \mathbf{E})$ is a graphical structure, or more precisely, a DAG, which codifies the qualitative knowledge. Nodes in $\boldsymbol{V} = \{X_1, X_2, \ldots, X_n\}$ represent the random variables [1] from the domain we are considering; and the structure 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).
- Θ is the set of numerical parameters which codifies the quantitative knowledge, usually conditional probability distributions drawn from the graph structure. For each variable $X_i \in \boldsymbol{V}$ there is 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

[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.

Condition:

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

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 [23].

We denote that variables in $\mathbf{X}$ are conditionally independent of variables in $\mathbf{Y}$ given the set $\mathbf{Z}$ in a DAG $\mathcal{G}$ (through d-separation) by $\langle \mathbf{X}, \mathbf{Y}|\mathbf{Z}\rangle_{\mathcal{G}}$. When such a independence is related to the probability distribution p, it 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 it has two incoming edges, i.e. a subgraph $A \rightarrow X \leftarrow B$ (also known as a head to head node). If the tail nodes of a collider node are not adjacent in G, i.e. A and B are not adjacent in G, this subgraph is called a *v-structure* in X.

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 holds:

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 Independence Map, *I-map*, of a probability distribution p if $\langle \mathbf{X}, \mathbf{Y}|\mathbf{Z}\rangle_{\mathcal{G}} \Rightarrow I_p(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$, and is *minimal* if no arc can be removed from $\mathcal{G}$ without violating the I-map condition. $\mathcal{G}$ is a Dependence Map, *D-map*, of p if $\langle \mathbf{X}, \mathbf{Y}|\mathbf{Z}\rangle_{\mathcal{G}} \Leftarrow I_p(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$.

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 will say that p and G are *faithful* to each other [1,17]. Furthermore, a distribution p is faithful if there exists a graph, G, to which it is faithful. In a faithful BN $\langle \mathbf{X}, \mathbf{Y}|\mathbf{Z}\rangle_{\mathcal{G}} \Leftrightarrow I_p(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$. We will assume faithfulness in the rest of the paper.

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 [22]. In general, the aim of learning Bayesian networks is to obtain a DAG that is a minimal I-map of the probability distribution encoded by the dataset.

Given two DAGs, Verma and Pearl [24] identified when two models are isomorphic or equivalent.

Theorem 1 [24]**.** *Let $\mathcal{G}$ and $\mathcal{H}$ denote two arbitrary DAGs. $\mathcal{G}$ and $\mathcal{H}$ are equivalent if and only if they have the same skeleton*[3] *and the same set of v-structures.*

In order to obtain a unique graphical representation of those DAGs representing the same set of conditional independence statements, i.e. an equivalence class, we need the definition of certain important concepts.

Definition 5. A partially directed acyclic graph (PDAG) (Fig. 1(b)) is a hybrid graph with no directed cycles, that contains both directed and undirected edges. In a PDAG, a pair of nodes X and Y are neighbors if they are connected by an undirected edge, and are adjacent if they are connected by a directed or undirected edge. A *semi-directed* path between two nodes A and B is a path including both directed and undirected edges, but in the case of directed edges, they must point to B.

We use $\varepsilon(\mathcal{P})$ to denote the equivalence class induced by a PDAG $\mathcal{P}$. The basis to identify an equivalence class of DAGs is a PDAG containing a directed edge for every edge participating in a v-structure and an undirected edge for every other edge. There may be many other PDAGs, however, which correspond to the same equivalence class. For example, any DAG interpreted as a PDAG can be used to represent its own equivalence class. Furthermore, there are PDAGs which do not correspond to an equivalence class.

Definition 6. If a DAG $\mathcal{G}$ has the same skeleton and the same set of v-structures as a PDAG $\mathcal{P}$, and if every directed edge in $\mathcal{P}$ is also in $\mathcal{G}$, then $\mathcal{G}$ is a consistent extension of $\mathcal{P}$.

[3] The skeleton of a DAG $\mathcal{G}$ is the graph created by converting the directed edges of $\mathcal{G}$ into undirected edges.

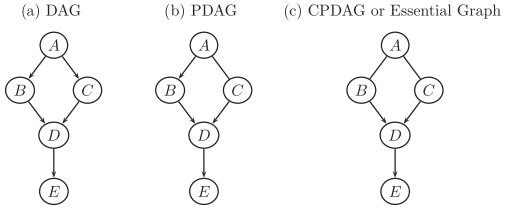

Fig. 1. DAG, PDAG and CPDAD or essential graph.

The DAG in Fig. 1(a) is a consistent extension of the PDAGs shown in Fig. 1(b) and (c). Any DAG which is a consistent extension of the PDAG $\mathcal{P}$ must also be contained in $\varepsilon(\mathcal{P})$, but not every DAG in $\varepsilon(\mathcal{P})$ is a consistent extension of $\mathcal{P}$. If there is at least one consistent extension of a PDAG $\mathcal{P}$, we say that $\mathcal{P}$ admits a consistent extension.

Definition 7. A directed edge $X \to Y$ is compelled in a DAG $\mathcal{G}$ if for every DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$, $X \to Y$ exists in $\mathcal{G}'$. For an edge $X \to Y$ in $\mathcal{G}$, if it is not compelled then it is reversible in $\mathcal{G}$; that is, there exists some DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$ in which $X \to Y$ has an opposite orientation.

Definition 8. In a *Completed PDAG* (CPDAG) $\mathcal{P}^c$, directed edges are *compelled* and they represent an edge that has the same orientation for every member of the equivalence class. Undirected edges represent *reversible edges*, i.e. edges that are not compelled.

For a CPDAG, $\mathcal{P}^c$, as opposed to an arbitrary PDAGs, every DAG contained in $\varepsilon(\mathcal{P}^c)$ is a consistent extension of $\mathcal{P}^c$.

Last, given an equivalence class of DAGs, the CPDAG representation is unique. Thus, CPDAGs are used to represent the space of equivalence classes, the *E-space*. Fig. 1(c) shows a CPDAG representing an equivalence class ε and Fig. 1(a) shows a DAG that belongs to ε.

3. Learning Bayesian networks

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 a DAG $\mathcal{G}^*$ such that

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

where $f(\mathcal{G} : D)$ is a scoring metric (or scoring criterion) 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 the DAGs with n nodes.

Although there are many methods and alternative search spaces to look for $\mathcal{G}^*$, the most used in practice are those based on local search algorithms and the space of Directed Acyclic Graphs (*D-space*). Efficient evaluation of neighbors of DAGs in these local search algorithms 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 the scores of the subgraphs formed by each node and its parents in $\mathcal{G}$. Formally, if f is decomposable, then:

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

If a decomposable metric is used, graphs resulting of changing one arc can be efficiently evaluated. Thus, this kind of (local) methods reuse the computations carried out at previous stages, and only the statistics corresponding to the variables whose parents have been modified need to be recomputed. The most used metrics in the literature related to BN learning are: the Bayesian Dirichlet equivalent (BDe) [5], the Minimum Description Length (MDL) [14] and the Bayesian Information Criterion (BIC) [15].

4. Asymptotic behavior of a scoring metric

In Section 3 it was mentioned that if a scoring metric is *decomposable* in the presence of full data, its use improves efficiency of some (local) learning algorithms. There are some other properties of scoring metrics which also make it possible to improve the learning process. These properties support the proposals presented in this work:

Definition 9. A scoring metric f is *score equivalent* if for any pair of equivalent DAGs, $\mathcal{G}$ and $\mathcal{G}'$, $f(\mathcal{G} : D) = f(\mathcal{G}' : D)$.

Definition 10 (Consistent scoring criterion [6])**.** 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:

Algorithm 1. GES

```
1: procedure GES
2:     ε ← Empty network (E = ∅)
3:     ε ← FES(ε)
4:     ε ← BES(ε)
5:     return ε
6: end procedure
```

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 $(\mathcal{G}, \Theta)$ exactly represents p. Of course, if two graphs represent the same set of conditional independence relationships, then the sparser one should receive more merit. This is the reason which supports looking for (I-map) sparser graphs.

Proposition 1 (From [6]). *The Bayesian Dirichlet equivalent (BDe) [5], Minimum Description Length (MDL) [14] and Bayesian Information Criterion (BIC) [15] metrics are score equivalent and consistent.*

Definition 11 (Locally consistent scoring criterion [6]). 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 our proposal is based on, as we will explain in Section 6, since it allows us to (asymptotically) assume that the differences computed by a locally consistent scoring metric f can be used as conditional independence tests over the dataset D. In order to do that, it is also necessary to assume that D constitutes a sample which is isomorphic[4] to a graph.

Proposition 2 [6]. *The BDe, MDL and BIC metrics are locally consistent.*

5. Greedy Equivalence Search (GES)

Unlike most BN learning algorithms, which are defined over the space of DAGs, GES [6] carries out the search for the best network in the space of equivalence classes. In order to do that, it takes as a starting point the equivalence class ε corresponding to the (unique) DAG with no edges, and carries out a two-stage greedy procedure. In the first phase, this procedure adds dependences to the model until it reaches a local maximum. Afterwards, in the second phase, dependences are removed (also by following a greedy approach) from such a model. The procedure is guaranteed to stop at a local maximum, which is the equivalence class returned as the solution.

The structure of GES is shown in Algorithm 1, and the pseudocode for each one of the two phases, FES (Forward Equivalence Search) and BES (Backward Equivalence Search), is detailed in Algorithms 2 and 3 respectively. At each iteration, the procedure adds (FES) or deletes (BES) an arc to the current ε equivalence class so as to maximize the score metric f. As the resulting PDAG may not be a CPDAG, it is necessary to transform it. The transformation algorithm which does that, *PDAGtoCPDAG*, has linear complexity, and is only called once per iteration, every time an arc is added or deleted.

In relation of FES, the selection of the best arc at each iteration is carried out by means of the operator *Insert*, which is specified in Function FS of the Algorithm 2. The formal definition of this operator is:

Definition 12 (*Insert*$(X, Y, \mathbf{T})$ [6, *Definition* 12]). For non-adjacent nodes X and Y in the current CPDAG $\mathcal{P}^{\mathcal{C}}$, and for any subset $\mathbf{T} \subseteq \mathbf{T_0}$ (neighbors of Y that are not adjacent to X), the Insert$(X, Y, \mathbf{T})$ operator modifies $\mathcal{P}^{\mathcal{C}}$ by (1) inserting the directed edge $X \rightarrow Y$; and (2) for each $T \in \mathbf{T}$, directing the previously undirected edge between T and Y as $T \rightarrow Y$.

So, besides choosing the inserted arc, the operation also stores a subset $\mathbf{T}$ for the best insertion (lines 7 and 8 of Algorithm 2). In line 5, the algorithm tests the validity of the arc, in order to assure that the resulting PDAG, once included, admits a consistent extension, and hence the *E-space* is transversed.

[4] In fact, [6] proves that the isomorphic condition can be relaxed.

Algorithm 2. FES

```
1: procedure FES(ε)
2:     repeat
3:         (X → Y, T) = FS(ε)
4:         ε = Apply Insert(X, Y, T) to ε
5:         ε = PDAGtoCPDAG(ε)
6:     until (X → Y = null)
7:     return ε
8: end procedure

9: function FS(ε)
10:    edge = null; best = 0
11:    for all X ∈ V do
12:        for all Y ∈ V | (Y ≠ X) ∧ Y is not adjacent to X do
13:            for all T ⊆ T0 | Test(X → Y, T) = true do
14:                Compute Δ = Insert(X, Y, T)
15:                if Δ > best then best = Δ
16:                    edge = (X → Y); subset = T
17:                end if
18:            end for
19:        end for
20:    end for
21:    return (X → Y, T) = (edge, subset)
22: end function
```

Algorithm 3. BES

```
1: procedure BES(ε)
2:     repeat
3:         (X → Y, H) = BS(ε)
4:         ε = Apply Delete(X, Y, T) to ε
5:         ε = PDAGtoCPDAG(ε)
6:     until (X → Y = null)
7:     return ε
8: end procedure

9: function BS(ε)
10:    edge = null; best = 0
11:    for all X ∈ V do
12:        for all Y ∈ V | (Y ≠ X) ∧ Y is adjacent to X do
13:            for all H ⊆ H0 | Test(X ↛ Y, H) = true do
14:                Compute Δ = Delete(X, Y, H)
15:                if Δ > best then best = Δ
16:                    edge = (X → Y), subset = H
17:                end if
18:            end for
19:        end for
20:    end for
21:    return (X → Y, H) = (edge, subset)
22: end function
```

Theorem 2 [6, Theorem 15]**.** *Let the $\mathcal{P}^{\mathcal{C}}$ be the current CPDAG, and let $\mathcal{P}^{\mathcal{C}'}$ be the CPDAG once we have inserted the link $X \to Y$ to $\mathcal{P}^{\mathcal{C}}$, with the Insert operation. The operation is valid if and only if:*

- *Let $\mathbf{NA}_{Y,X}$ be the subset of nodes that are neighbors to Y and adjacents to X, then $\mathbf{NA}_{Y,X} \cup \mathbf{T}$ is a clique.*[5]
- *Every semi-directed path from X to Y contains a node in $\mathbf{NA}_{Y,X} \cup \mathbf{T}$.*

Given a metric f with the properties detailed in Section 4, the increment of the scoring metric produced by the insertion operation is computed in line 14 of Algorithm 2 (FES). The way such an increment is calculated is determined by the following corollary of the above theorem:

Corollary 1 [6, Corollary 16]**.** *The increase in score that results from applying a valid operator Insert(X, Y, $\mathbf{T}$) to a CPDAG $\mathcal{P}^{\mathcal{C}}$ is:*

$$f(Y, Pa_{\mathcal{P}^{\mathcal{C}}}(Y) \cup \{X\} \cup \mathbf{T} \cup \mathbf{NA}_{Y,X}) - f(Y, Pa_{\mathcal{P}^{\mathcal{C}}}(Y) \cup \mathbf{T} \cup \mathbf{NA}_{Y,X}).$$

Once FES has reached a local optimum, this equivalence class is taken as a starting point for BES, the second phase of GES, which iteratively removes arcs also guided by a greedy criterion. The pseudocode with the specification of BES is shown in

[5] A complete subgraph.

Algorithm 3. As can be seen, the removal of arcs is carried out by means of the operator *Delete*. The formal definition of such an operator is as follows:

Definition 13 (*Delete*(X, Y, $\mathbf{H}$))**.** For adjacent nodes X and Y in the current CPDAG $\mathcal{P}^{\mathcal{C}}$, connected as $X - Y$ or $X \rightarrow Y$, and for any subset $\mathbf{H}$ of neighbors of Y which are adjacent to X, the Delete(X, Y, $\mathbf{H}$) operator modifies $\mathcal{P}^{\mathcal{C}}$ by (1) deleting the edge between X and Y; (2) converting the previously undirected edge between H and Y into a directed one, $Y \rightarrow H$, for each $H \in \mathbf{H}$; and (3), converting any previously undirected edge between H and X into a directed one $X \rightarrow H$.

Again the best operation is selected by using a scoring metric f, and subset $\mathbf{H}$ must be saved (lines 7 and 8 of Algorithm 3). In order to assure the correct movement, it is also necessary to test the validity of the arc (line 5).

Theorem 3 [6, Theorem 17]**.** *Let $\mathcal{P}^{\mathcal{C}}$ be the current CPDAG, and let $\mathcal{P}^{\mathcal{C}'}$ be the state once we have deleted the edge $X \rightarrow Y$ or $X - Y$. The Delete operation is valid if and only if $\mathbf{NA}_{Y,X} \setminus \mathbf{H}$ is a clique.*

The increment in the scoring metric produced by the application of the *Delete* operation to the current PDAG is carried out in line 14 of Algorithm 3, while determining which arc must be removed. The following result shows the way such an increment, Δ, is computed.

Corollary 2 [6, Corollary 18]**.** *The increase in score that results from applying a valid operator Delete(X, Y, $\mathbf{H}$) to a CPDAG $\mathcal{P}^{\mathcal{C}}$ is:*

$$f(Y, Pa_{\mathcal{P}^{\mathcal{C}}}(Y) \cup \{\mathbf{NA}_{Y,X} \setminus \mathbf{H}\} \setminus X) - f(Y, Pa_{\mathcal{P}^{\mathcal{C}}}(Y) \cup \{\mathbf{NA}_{Y,X} \setminus \mathbf{H}\}).$$

Lemmas 9 and 10 in [6] show that the equivalence class obtained at the end of FES is, asymptotically, an I-map of p, whereas the equivalence class obtained at the end of BES, and hence of GES, is asymptotically a perfect map of p. Note that faithfulness is required in order to assure these theoretical properties. If data is not sampled from a perfectly Markovian distribution, GES can fail to recover the asymptotically optimal Bayesian network. In order to solve this problem, some alternative representations of BN structure have been recently proposed, such as the *standard imsets* (vectors of integers) [25] or the *characteristic imsets* (zero–one vectors) [26]. However, such alternative representations are still in a preliminary state [26,27].

From Algorithm 2, it holds that the number of operations carried out by FES is exponential on the size of the set $\mathbf{T_0}$. For example, let us consider a dataset sampled from a Naive Bayes network with a class variable C and n predictive variables $X_1, \ldots, X_n$. At iteration n of FES, the algorithm has correctly identified $n - 1$ arcs. Now, the algorithm needs to evaluate the operations Insert(C, X_i, $\emptyset$)) and Insert(X_i, C, $\mathbf{T}$) ($\forall T \subseteq \{X_1, \ldots, X_n\} \setminus \{X_i\}$)). The last operation implies the evaluation of 2^{n-1} different subsets.

This kind of structures are not rare and, in fact, appear in real networks such as, for example, the Hailfinder (see Section 7.2.1). Therefore, the overall running-time of GES is predominated by the first constructive phase (FES). In order to provide some results for this, we performed a simple experiment with synthetic databases sampled from randomly generated Bayesian networks (see Section 7 for further details) and we concluded that the time spent by the algorithm in the BES phase is not relevant if compared with that spent in the FES phase.

6. Scaling up GES

In this section we propose some variants of the original GES algorithm. The idea behind these algorithms is to modify the first phase of GES in order to reduce the number of operations computed. The descendant phase (BES) will not be modified; we are only going to propose some modifications to FES. The reason to maintain the original descendant phase is that, to our knowledge, it is not possible to simplify this search and still guarantee the theoretical properties of GES. Moreover, in practice, the number of iterations carried out by BES is far smaller than that carried out by FES (see Section 7).

Our strategy is to assure that the proposed modified versions of FES retain the same theoretical properties as the original FES, i.e. they return a minimal I-map, or at least, an I-map of the generative distribution under the faithfulness assumption. If this behavior is assured, then adding the original descendant phase of GES, BES, to our algorithms should lead (asymptotically) to the same result as the original GES (Lemmas 9 and 10 [6]).

Firstly, we propose, in Section 6.1, to restrict the search dynamically during the inclusion of edges if we find that two nodes are conditionally independent. Secondly, in Section 6.2, we propose to replace the exhaustive search of the subset $\mathbf{T}$ by a greedy one. Finally, we propose several combinations of these two main ideas in Section 6.3.

6.1. Constrained FES: FES_C, FES_{C^*} and FES_{iC}

In these algorithms we apply a similar approach to the progressive and dynamic restriction of the neighborhood used in [21]. The idea behind the algorithm is to reduce the number of operations, by restricting the pairs of variables X and Y for which the Insert(X, Y, $\mathbf{T}$) operation is tested. In order to do that, we maintain a set of forbidden parents for each variable X_i,

Algorithm 4. FES_C

```
1: procedure FES_C(ε)
2:     for all X ∈ V do FP(X) = ∅
3:     end for
4:     repeat
5:         (X → Y, T) = FS_C(ε)
6:         ε = Apply Insert(X, Y, T) to ε
7:         ε = PDAGtoCPDAG(ε)
8:     until (X → Y = null)
9:     return ε
10: end procedure

11: function FS_C(ε)
12:     edge = null; best = 0
13:     for all X ∈ V do
14:         for all Y ∈ V | (Y ≠ X) ∧ (X ∉ FP(Y)) ∧ Y is not adjacent to X do
15:             fp = true
16:             for all T ⊆ T_0 | Test(X → Y, T) = true do
17:                 Compute Δ = Insert(X, Y, T)
18:                 if (Δ > 0) then fp = false
19:                 end if
20:                 if (Δ > best) then
21:                     best = Δ; edge = (X → Y); subset = T
22:                 end if
23:             end for
24:             if fp then FP(Y) = FP(Y) ∪ X
25:             end if
26:         end for
27:     end for
28:     return (X → Y, T) = (edge, subset)
29: end function
```

Algorithm 5. FES_{C^*}

```
1: procedure FES_{C*}(ε)
2:     ε = FES_C(ε)
3:     ε = FES(ε)
4:     return ε
5: end procedure
```

denoted as $FP(X_i)$. The difference between this algorithm and FES is that the definition of the neighborhood of ε is slightly modified.

The basic modification is called FES_C and its pseudocode is shown in Algorithm 4. In this algorithm, the neighborhood is obtained from ε by computing, for all variables X and all the non-adjacent variables Y such that $X \notin FP(Y)$, the operations Insert(X, Y, $\mathbf{T}$) (line 14 of Algorithm 4). A variable X is added to the forbidden parents list of another variable Y only if the evaluation of every operation Insert(X, Y, $\mathbf{T}$) ($\forall \mathbf{T} \subseteq \mathbf{T_0}$) results in a network that is worse than the previous one (line 24 of Algorithm 4). In this situation, according to the definition of local consistency (def. 11), we have found a subset $\mathbf{S}$ such that X and Y are conditionally independent given such a subset $\mathbf{S} = \mathbf{T} \cup \mathbf{NA}_{Y,X}$ (see Corollary 1).

Unfortunately, we cannot assure that the result of FES_C (Algorithm 4) is an I-map.[6] In order to correct this problem, we propose two iterated versions of this algorithm, denoted as FES_{C^*} and FES_{iC}.

The first version (Algorithm 5) is straightforward, and reduced to using the output of FES_C as the starting point of a second ascendant phase in which (original unrestricted) FES (Algorithm 2) is carried out. That is, the algorithm performs two forward stages, the first one using FES_C and the second one using FES. With this proposal, if the result of the first iteration is good enough, then the second one should iterate only a few times.

Proposition 3. *Let ε denote the equivalence class returned by FES_{C^*} and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. The proof is trivial because the last step of the algorithm runs FES and it is known that FES reaches an I-map regardless of the starting point [6]. □

The second iterated algorithm, called FES_{iC} (Algorithm 6), runs FES_C iteratively, using the equivalence class returned by the previous iteration as the initial solution, until no improvement is achieved. As we can observe, FES_C starts by emptying the forbidden parent lists, so at the beginning of each iteration of FES_{iC} all possible edge additions are considered.

[6] In fact, we have found neither evidence to confirm this last sentence nor a counterexample. In any case, the effort we have to make in order to prove the correctness does not compensate for the empirical results that this algorithm offers in our experiments.

Algorithm 6. FES_{iC}

```
1: procedure FES_iC(ε)
2:     repeat
3:         ε = FES_C(ε)
4:     until (ε does not change)
5:     return ε
6: end procedure
```

Algorithm 7. FES_{G}

```
1: procedure FES_G(ε)
2:     repeat
3:         (X → Y, T) = FS_G(ε)
4:         ε = Apply Insert(X, Y, T) to ε
5:         ε = PDAGtoCPDAG(ε)
6:     until (X → Y = null)
7:     return ε
8: end procedure

9: function FS_G(ε)
10:     edge = null; subset = null; bestEdge = 0
11:     for all X ∈ V do
12:         for all Y ∈ V | (Y ≠ X) ∧ Y is not adjacent to X ∧ Test(X → Y, ∅) = true do
13:             T_1 = ∅; best = 0; ok = false
14:             Compute Δ = Insert(X, Y, T_1)
15:             if (Δ > best) then
16:                 best = Δ; ok = true
17:             end if
18:             while ok do
19:                 ok = false
20:                 for all T ∈ (T_0 \ T_1) | Test(X → Y, T_1 ∪ T)) = true do
21:                     Compute Δ = Insert(X, Y, T_1 ∪ T))
22:                     if (Δ > best) then
23:                         best = Δ, T_2 = T_1 ∪ T
24:                     end if
25:                 end for
26:                 if (best > Insert(X, Y, T_1)) then
27:                     T_1 = T_2; ok = true
28:                 end if
29:             end while
30:             if (best > bestEdge) then bestEdge = best
31:                 edge = (X → Y); subset = T_1
32:             end if
33:         end for
34:     end for
35:     return (X → Y, T) = (edge, subset)
36: end function
```

Proposition 4. *Let ε denote the equivalence class returned by FES_{iC} and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. In the last iteration of the algorithm, no edge is added. As it starts with all the sets of forbidden parents empty, it behaves like the original FES, and therefore we can be sure the algorithm has reached a local optimum (none of all the possible edge additions to ε increases the score) and this local maximum contains p [6]. □

6.2. Greedy FES: FES_G and FES_{G^*}

As was shown in Section 5, GES tests the operation Insert(X, Y, $\mathbf{T}$) for all $\mathbf{T} \subseteq \mathbf{T_0}$. For a set $\mathbf{T_0}$ of size k, 2^k operations need to be computed. In practice, to overcome this issue, a maximum number of parents (maximum size of $\mathbf{T}$) is set a priori. However, this parameter is difficult to manage in general and its use may degrade the theoretical properties of the algorithm.

Our proposal is to reduce the candidate neighbors by, instead of performing an exhaustive search over all possible subsets of $\mathbf{T_0}$, performing a greedy search in order to find a reasonably good subset of $\mathbf{T_0}$. The proposed algorithm, FES_G, uses a restricted subset of the ε neighborhood, where the members are computed using a greedy procedure.

Algorithm 7 shows the FES_G algorithm. For each pair of variables X and Y, it first initializes $\mathbf{T}_1 = \emptyset$ and checks and computes the score of the operation Insert(X, Y, $\mathbf{T}_1$). If this operation increases the score of the previous network, it checks and scores the operations Insert(X, Y, $\mathbf{T_2} = \mathbf{T_1} \cup \{T\}$) ($\forall T \in \mathbf{T_0} \setminus \mathbf{T_1}$). The algorithm iteratively replaces $\mathbf{T}_1$ with the best $\mathbf{T_2}$ and it repeats this last step until no such replacement increases the score or when $\mathbf{T_1} = \mathbf{T_0}$.

Proposition 5. *Let ε denote the equivalence class returned by FES_G and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. Let us suppose the contrary, i.e. ε does not contain p. Then there exists at least two variables X_i and X_j such that $\langle X_i, X_j | Pa_\varepsilon(X_i)\rangle_\varepsilon$ and $\neg I_p(X_i, X_j | Pa_\varepsilon(X_i))$. Thus, ε cannot be a local optimum because the addition of the arc $X_j \rightarrow X_i$ has a positive difference, with, at least, the empty set $\mathbf{T} = \emptyset$. □

Finally, although FES_G guarantees the I-map condition under faithfulness, in order to achieve better results with real (limited) data, a new algorithm is proposed. The algorithm is called FES_{G^*} and it is quite similar to FES_{C^*}. The only difference between FES_{G^*} and FES_{C^*} is that FES_{G^*} uses FES_G in the first iteration instead of FES_C.

Proposition 6. *Let ε denote the equivalence class returned by FES_{G^*} and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. See proof of Proposition 3. □

6.3. Constrained greedy FES: FES_{CG}, FES_{CG^*}, FES_{iCG}

In this section we propose the last family of algorithms, which are derived from the two previous proposals. The first one, denoted as FES_{CG}, combines both the constrained and the greedy approaches, i.e. it performs a greedy search over the space of candidate neighbors (as FES_G does), but it also uses a list of forbidden parents to constrain the search (as FES_C does). The difference between this algorithm and FES_G is the use of a set of forbidden parents for each variable. This set is reset at the beginning. Then, the algorithm iteratively runs an auxiliary function, FS_{CG}, which is similar to FS_G. There is a difference between the two auxiliary functions, since the first one only checks an operation of type Insert(X,Y,**T**) if $X \notin FP(Y)$, and a variable X is added to the list of forbidden parents of another variable Y if the operation Insert(X, Y, $\emptyset$) decreases the score of the network. As happened with FES_C, it is not guaranteed that the algorithm will return an I-map.

In order to solve this issue, we propose an iterated version of the previous algorithm, denoted as FES_{iCG}. This algorithm simply iterates FES_{CG} until no improvement is achieved.

Proposition 7. *Let ε denote the equivalence class that results at the end of the first phase of FES_{iCG} and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. In the last iteration of the FES_{iCG} algorithm, all the sets of forbidden parents are empty. In this situation, the behavior of the algorithm would be the same as the behavior of FES_G. □

The last variant we consider is an iterated algorithm that runs two iterations. The first iteration executes FES_{CG}, whereas the second iteration, which starts with the resulting equivalence class returned by the previous iteration, executes FES. This last algorithm is called FES_{CG^*}.

Proposition 8. *Let ε denote the equivalence class returned by FES_{CG^*} and p the generative probability distribution faithful to a DAG. Then, asymptotically, ε contains p.*

Proof. See proof of Proposition 3. □

7. Experimental evaluation

7.1. Experimentation setup

In this section, we perform an empirical evaluation of the proposed algorithms. Although in the previous sections we have only focused on the forward phase, all proposed algorithms execute, as GES does, both a forward and a backward phase. In all cases, we have used the original unmodified BES algorithm. A summary of the main characteristics of the algorithms proposed is shown in Table 1.

The reasons for using BES are twofold: (a) the algorithm guarantees that, under faithfulness assumptions, it obtains a perfect map of the original distribution if the initial network is an I-map; and (b) in practice, BES is only executed during few iterations, and therefore its contribution to the runtime of the algorithm is much lower than the contribution of FES. In order to support this last affirmation we performed the following experiment. We ran the GES algorithm with 12 different types of databases (see Section 7.3 for details) and collected the average number of iterations of both the FES and BES phases. Fig. 2 shows the results obtained in this experiment. As it is shown in the graph, BES runs only a few iterations while FES runs several hundred of them.

Table 1
Algorithms used in the experimental evaluation.

	Forward phase	Search of X and Y	Search of **T**
GES	FES	Exhaustive	Exhaustive
GES_C	FES_C	Constrained	Exhaustive
GES_{iC}	n x FES_C	Constrained	Exhaustive
GES_{C*}	FES_C + FES	Constrained + Exhaustive	Exhaustive
GES_G	FES_G	Exhaustive	Greedy
GES_{G*}	FES_G + FES	Exhaustive	Greedy + Exhaustive
GES_{CG}	FES_{CG}	Constrained	Greedy
GES_{iCG}	n x FES_{CG}	Constrained	Greedy
GES_{CG*}	FES_{CG} + FES	Constrained + Exhaustive	Greedy + Exhaustive

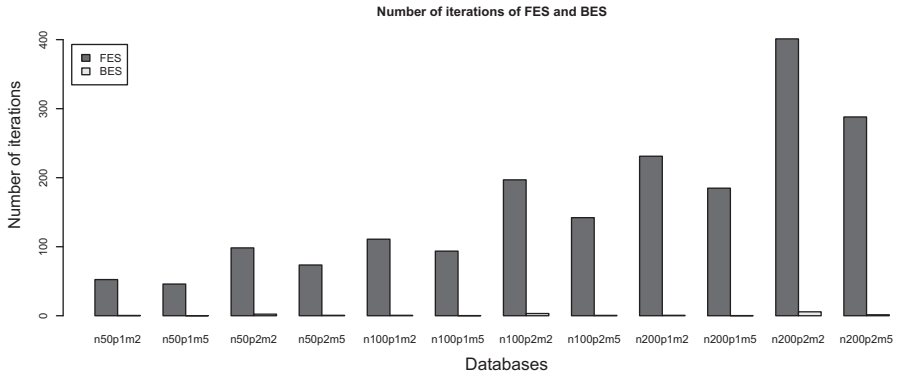

Fig. 2. Number of iterations of FES and GES.

7.1.1. Implementation and running environment

The algorithms are coded in Java and are based on the open implementation included in `Tetrad 4.3.9`.[7] The implementation includes the optimizations suggested in [6]. However, the maximum number of parents is not limited. It is important to remark that our proposals are aimed at improving the efficiency of GES. Therefore, all the algorithms use the same data structures and share as much code as possible.

The score metric used in the algorithms is the Bayesian Dirichlet equivalent in its uniform prior version BDeu [5]; The equivalent sample size (N') used in the experiments is 10; and the network priors are calculated as in [8], where the kappa parameter is $\kappa = 1/(N' + 1)$. Then, the BDeu score used in the experiments is given by the following equation:

$$BDeu(\mathcal{G}, D) = \log\left(\prod_{i=1}^{n}\left(\frac{1}{10+1}\right)^{(r_i-1)q_i}\prod_{j=1}^{q_i}\frac{\Gamma(\frac{10}{q_i})}{\Gamma(N_{ij}+\frac{10}{q_i})}\cdot\prod_{k=1}^{r_i}\frac{\Gamma(N_{ijk}+\frac{10}{r_iq_i})}{\Gamma(\frac{10}{r_iq_i})}\right)$$

for a dataset D with n discrete variables, each variable having r_i values and q_i being the number of configurations for the parent set of x_i. In this score, $\Gamma(\cdot)$ is the *Gamma* function, N_{ijk} is the number of cases in database D in which variable $x_i = k$ for the j-th configuration of its parents, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

The algorithms run on in a cluster of computers. Each node of the cluster is composed of two quad-core Intel Xenon at 3.00 GHz and 32 GB of RAM. We run up to 4 executions at the same time in each node and the maximum heap size[8] of each execution was set to 8 GB.

7.1.2. Performance indicators

In order to compare the algorithms, we consider two kinds of factors as performance indicators: the quality of the network obtained by the algorithms, and the complexity of each algorithm.

The quality of the networks obtained by the algorithms is given by both its BDeu score and its Structural Hamming Distance (SHD) with respect to the original network. The SHD is defined in CPDAGs, and is computed as the number of the following operators required to make the CPDAGs match: add or delete an undirected edge; and add, remove, or reverse the orientation of an edge.

The complexity of the algorithms is given by the following factors: (a) the execution time of the algorithms; (b) the number of computations of the score metric; (c) the maximum numbers of subsets **T** checked in order to compute an operation $X \to Y$; and (d) the size of the biggest CPT created in the scoring function.

[7] http://www.phil.cmu.edu/projects/tetrad/.
[8] Memory used by the Java Virtual Machine to store dynamically created objects.

Table 2
BNs used in the experiments.

Name	# of vars	# of edges	Max. parents	Max. children	Max. neighbors (CPDAG)
Alarm	37	46	4	5	1
Barley	48	84	4	5	3
Hailfinder	56	66	4	16	17
Insurance	27	52	3	7	8
Mildew	35	46	3	3	0
Munin1	189	282	3	15	4
Pigs	441	592	2	39	0

Table 3
BDeu score of the networks.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	**−49534.9**	−49567.8	**−49534.9**	**−49534.9**	**−49534.9**
Barley	**−286061.9**	−287657.3	**−286061.9**	**−286061.9**	**−286061.9**
HailFinder	−268446.1*	−268446.1*	−268446.1*	−268446.1*	**−253707.5**
Insurance	**−68579.5**	−68617.4	**−68579.5**	**−68579.5**	−68614.3
Mildew	**−260535.7**	**−260535.7**	**−260535.7**	**−260535.7**	**−260535.7**
Munin1	−229149.7	−230341.2	−229729.2	−229728.1	−229261.8
Pigs	**−1684624.8**	**−1684624.8**	**−1684624.8**	**−1684624.8**	**−1684624.8**

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	**−49534.9**	−49567.8	**−49534.9**	**−49534.9**
Barley	**−286061.9**	−287657.3	**−286061.9**	**−286061.9**
HailFinder	**−253707.5**	−253711.5	**−253707.5**	**−253707.5**
Insurance	−68591.7	−68650.1	−68612.3	−68596.2
Mildew	**−260535.7**	**−260535.7**	**−260535.7**	**−260535.7**
Munin1	**−229148.3**	−229784.5	−229267.5	−229730.3
Pigs	**−1684624.8**	**−1684624.8**	**−1684624.8**	**−1684624.8**

Although the execution time is the most important factor when evaluating the time complexity of the algorithm, it depends on both the implementation and the specifications of the computer on which the algorithm is executed. On the other hand, the number of calls to the scoring metric is independent of the implementation, and it is also a good indicator of the time complexity of the algorithms. Finally, the maximum number of subsets evaluated gives us an idea of the reduction in evaluation time that some algorithms achieve, specially when complex structures are present in the networks.

Regarding the memory requirements, there are mainly two data structures that have variable memory requirements. The first one is the cache memory used to store the score of the substructures evaluated. The number of elements in this cache memory is given by the number of computations of the metric. The second one is the CPT that is created in order to store the counts N_{ijk} inside the scoring function. Although these structures are deleted after the score is calculated, we have checked in the experiments that the biggest memory requirements are produced when a big CPT has to be built.

Due to the complexity of the experiments, some executions end abruptly with an error for surpassing the maximum heap size of 8Gb (an example of such behaviour is shown in Section 7.2.1.). In these cases, which have been marked with an asterisk, we take as output the best network found by the algorithm up to that moment. Besides, in order to make it easier the interpretation of the tables, the best result obtained with each network have been highlighted with boldface.

7.2. Real-world network experiments

The first set of experiments were carried out by using real networks that are commonly referenced in the literature, all of them available at the Bayesian Networks Repository.[9] The selected networks are: Alarm [28], Barley [29], Hailfinder [30], Insurance [31], Mildew [32], Munin (version 1) [33] and Pigs [34]. Table 2 shows their main characteristics. As can be seen, some of them represent domains with few variables, like Alarm and Insurance, whereas others model complex domains with hundreds of variables, like Munin (version 1) or Pigs. For each one of the networks we sample five datasets of 5000 instances. The reported results for each network are the average of the results obtained with each dataset.

Tables 3 and 4 show the BDeu and the SHD results respectively. The results highlighted in bold are the best for the corresponding network. Results marked with an asterisk represent those executions that have finished with an error for requiring memory over the maximum heap size of 8 GB. When such errors occur, the algorithm returns the best network found up to that moment. In addition to the numerical results, it is important to remark the fact that, when two algorithms obtained the same result in one of the datasets, they also obtained the same result in the other four datasets of the same network.

We can draw some important conclusions from the results related to BDeu. A quick look shows that the differences between the algorithms are, in general, minimal. For instance, GES_{iC} and GES_{C*} obtain the same results as GES in 6 out of 7 networks. GES_G, GES_{CG} and GES_{iCG} also obtain good BDeu results, being equal to those obtained by GES in 4 cases, and

[9] http://www.cs.huji.ac.il/labs/compbio/Repository/.

Table 4
Structural difference with the original network.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	10.6	**10.0**	10.6	10.6	10.6
Barley	**63.4**	64.4	**63.4**	**63.4**	**63.4**
HailFinder	29.2*	28.2*	29.2*	29.2*	28.6
Insurance	50.6	50.6	50.6	50.6	35.2
Mildew	**36.0**	**36.0**	**36.0**	**36.0**	**36.0**
Munin1	229.6	219.0	220.4	220.4	233.0
Pigs	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	10.6	**10.0**	10.6	10.6
Barley	**63.4**	64.4	**63.4**	**63.4**
HailFinder	29.2	**28.0**	29.0	29.4
Insurance	35.2	**35.0**	35.2	35.2
Mildew	**36.0**	**36.0**	**36.0**	**36.0**
Munin1	229.4	221.2	222.4	**218.6**
Pigs	**0.0**	**0.0**	**0.0**	**0.0**

Table 5
Learning time.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	8.8	4.4	5.5	5.3	6.2
Barley	221.9	123.0	201.3	170.3	154.0
HailFinder	5 801.2*	1 996.9*	1 991.2*	1 876.6*	34.3
Insurance	5.8	3.3	4.3	4.1	3.7
Mildew	260.6	192.9	195.6	183.2	34.3
Munin1	71 630.0	58 063.2	64 147.4	62 521.6	41 444.4
Pigs	14 205.3	910.4	1 030.8	1 042.3	8 516.5

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	6.6	4.0	4.8	4.8
Barley	183.4	109.5	171.8	156.6
HailFinder	34.6	24.0	24.6	25.0
Insurance	4.4	2.9	3.4	3.8
Mildew	36.6	15.2	16.9	17.6
Munin1	44 142.6	57 412.5	66 156.9	63 144.2
Pigs	8 651.7	979.9	1 085.4	1 110.2

Table 6
Number of computations of the metric.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	5 561.8	3 109.8	4 084.4	4 084.4	3 154.6
Barley	8 757.4	3 347.6	7 036.2	7 036.2	4 198.0
HailFinder	54 090.0*	16 435.4*	16 435.4*	16 435.4*	5 943.0
Insurance	3 287.2	1 731.6	2 615.6	2 615.6	1 817.2
Mildew	3 561.4	1 762.8	2 811.2	2 811.2	2 087.2
Munin1	149 878.8	84 078.4	106 704.0	106 698.4	85 603.4
Pigs	549 143.2	226 456.0	345 375.4	345 375.4	375 625.2

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	3 519.2	3 147.6	3 147.6	3 272.4
Barley	7 178.0	4 204.2	4 204.2	6 921.2
HailFinder	5 943.0	5 943.0	5 943.0	5 718.2
Insurance	2 609.0	1 822.6	1 822.6	2 514.2
Mildew	2 567.2	2 086.2	2 086.2	2 540.2
Munin1	90 146.4	85 463.0	85 463.0	85 574.6
Pigs	375 800.0	375 632.0	375 632.0	345 367.8

better in the case of the Hailfinder network. It is also worth pointing out that GES_{G*} obtains the best BDeu score in 6 of the networks, and no other algorithm obtains the same result in the Munin network. Finally, GES_C and GES_{CG} are the algorithms which obtain the worst BDeu results. In relation to the SHD results, the first impression is that differences are also minimal. However, a better BDeu result does not imply a better SHD result. For example, the GES_{CG} algorithm obtains competitive SHD results although its BDeu results are not so good.

Finally, there are two networks that deserve a special analysis: Munin1 and Hailfinder. The first one is very complex, and has a lot of local optimal solutions. In this network, each algorithm obtains a different solution, all of them with a similar BDeu score. Possibly, the most remarkable thing regarding this network is that the best results, in terms of the BDeu score, correspond to networks with an SHD of around 230, whereas the worst results in the BDeu score correspond to networks with an SHD of around 220. The second one, the HailFinder network, is analyzed in detail in Section 7.2.1.

Table 7
Size of the biggest CPT.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	3 624.0	3 624.0	3 624.0	3 624.0	486.4
Barley	945 504.0	141 120.0	945 504.0	945 504.0	90 048.0
HailFinder	1 373 847 552.0*	1 287 982 080.0*	1 287 982 080.0*	1 287 982 080.0*	2 028.4
Insurance	1 856.0	1 420.8	1 740.8	1 740.8	400.0
Mildew	140 052 347.4	140 052 347.4	140 052 347.4	140 052 347.4	49 554.6
Munin1	2 313 360.0	2 220 825.6	2 220 825.6	2 220 825.6	8 223.6
Pigs	113.4	113.4	113.4	113.4	81.0

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	588.8	486.4	486.4	640.0
Barley	945 504.0	90 048.0	90 048.0	945 504.0
HailFinder	2 028.4	2 028.4	2 028.4	2 028.4
Insurance	1 600.0	400.0	400.0	1 600.0
Mildew	49 554.6	49 554.6	49 554.6	49 554.6
Munin1	9 156.0	8 223.6	8 223.6	9 744.0
Pigs	113.4	81.0	81.0	113.4

Table 8
Maximum numbers of neighbors visited to compute one insert operation.

Network	GES	GES_C	GES_{iC}	GES_{C*}	GES_G
Alarm	32.0	32.0	32.0	32.0	7.2
Barley	16.0	16.0	16.0	16.0	6.0
HailFinder	2 048.0*	2 048.0*	2 048.0*	2 048.0*	16.0
Insurance	14.4	14.4	14.4	14.4	7.2
Mildew	16.0	16.0	16.0	16.0	5.0
Munin1	57.6	57.6	57.6	57.6	10.0
Pigs	2.4	2.4	2.4	2.4	2.4

	GES_{G*}	GES_{CG}	GES_{iCG}	GES_{CG*}
Alarm	7.2	7.2	7.2	7.2
Barley	16.0	6.0	6.0	16.0
HailFinder	16.0	16.0	16.0	16.0
Insurance	16.0	7.2	7.2	14.4
Mildew	5.0	5.0	5.0	5.0
Munin1	12.8	10.0	10.0	14.4
Pigs	2.4	2.4	2.4	2.4

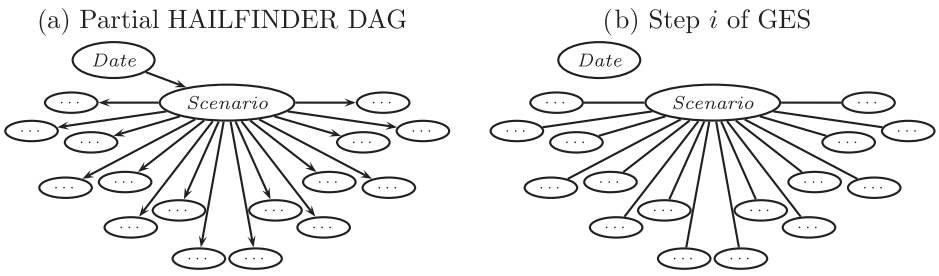

Fig. 3. (a) Partial HAILFINDER DAG. The graph was obtained by removing variables not in the Markov blanket of variable *Scenario* from the complete network. (b) Example of possible partial CPDAG obtained in step *i* of GES.

Tables 5–8 show the results relative to the complexity of the algorithms. As can be seen, all of the proposed algorithms are, in general, faster than GES, and they perform less computations of the metric. Regarding the size of the biggest CPT, and the maximum neighbors visited to compute one insert operation, the algorithms that perform a greedy forward phase obtain a considerable reduction in these factors, while the other algorithms obtain almost the same results as GES. From these latter experiments we can conclude that searching the set of neighbors greedily made the algorithms require less memory and, therefore, could be a better option when the availability of main memory is bounded.

7.2.1. The Hailfinder network

Hailfinder is not a complex network. However, the behavior of the algorithms when trying to learn it deserves a detailed analysis. Fig. 3(a) shows the structure of the HailFinder network [30] around the variable *Scenario*. Fig. 3(b) shows a possible equivalence class ε found in the step *i* of GES. The algorithm has to compute the neighborhood of ε. Thus, it has to compute the operation Insert(*Date*, *Scenario*, **T**) for each subset **T** of $\mathbf{T_0}$, $\mathbf{T_0}$ being the set of children of variable *Scenario* in the real network. As the set $\mathbf{T_0}$ has 16 variables, it has to compute 2^{16} different operations.

The previous example illustrates why the exhaustive search of **T** in the HailFinder network takes a considerable amount of time. However, the main problem with this network is not related with execution time but memory requirements since,

Table 9
Random BNs generated by the stochastic algorithm.

Name	# of vars	# of edges	Max. parents	Max. children	Max. neighbors (CPDAG)
n50p1m2	50	48.6 (37-59)	4.4 (3-6)	4.3 (3-7)	2.8 (1-5)
n50p1m5	50	48.6 (39-59)	4.4 (3-6)	4.2 (3-7)	2.8 (1-5)
n50p2m2	50	98.3 (84-118)	6.8 (5-10)	6.7 (5-8)	2.7 (2-4)
n50p2m5	50	97.6 (84-118)	6.6 (5-9)	6.7 (5-8)	2.7 (2-4)
n100p1m2	100	99.6 (81-121)	5.0 (4-8)	4.6 (3-6)	3.3 (2-4)
n100p1m5	100	98.6 (81-116)	4.9 (4-7)	4.6 (3-7)	3.2 (2-4)
n100p2m2	100	197.5 (175-222)	7.8 (6-10)	7.4 (5-9)	2.9 (2-4)
n100p2m5	100	197.5 (175-221)	7.9 (6-10)	7.6 (5-11)	3.0 (2-4)
n200p1m2	200	190.7 (159-226)	5.4 (4-7)	5.6 (4-8)	3.8 (3-6)
n200p1m5	200	191.6 (159-226)	5.3 (4-7)	5.4 (4-8)	3.7 (3-6)
n200p2m2	200	392.0 (358-450)	8.2 (7-10)	8.7 (6-11)	3.5 (2-6)
n200p2m5	200	393.0 (358-450)	8.4 (7-11)	8.5 (6-11)	3.4 (2-6)

when $\mathbf{T} = \mathbf{T_0}$, it becomes necessary to generate a CPT of size $\prod_{X_i \in \mathbf{V}} r_i(X_i) = 8.8 \cdot 10^{11}$, $\mathbf{V}$ being the set containing the 18 variables involved in the operation and $r_i(X_i)$ the number of values of X_i, in order to compute the score of the network.

The observation made in the previous paragraph is the reason why, under our settings, only the algorithms which perform the search of $\mathbf{T}$ greedily were able to recover the Hailfinder network. The algorithms that perform the search exhaustively ended in error due to the impossibility of creating CPT tables of more than 1 billion elements, as is shown in Table 7.

7.3. Synthetic network experiments

The experiments carried out in the previous section allow us to draw sound conclusions. However, we wanted to perform a more extensive experimentation. In order to do this, we used synthetic databases obtained from randomly generated models with different degrees of difficulty, taking into account the number of variables, the number of parents and the number of states for each variable. The procedure is the same as that followed in the previous section except that: (1) in this case, the GES_C and GES_{CG} algorithms are not included in the study, since in the previous experiments they obtained the worst results, moreover, they are not asymptotically optimal; and (2) we included the use of a hypothesis test to validate our conclusions.

As mentioned above the data generation procedure depends on 3 parameters: n, the number of variables; p, the average number of adjacencies in the network; and m, the maximum number of states for the variables. After setting the parameters, the following procedure is executed:

- Generation of the structure $\mathcal{G}$
  1. The graph $\mathcal{G}$ is initialized as an empty graph with n variables.
  2. For each pair of distinct variables X_i and X_j a random number r from a Bernoulli distribution with probability of occurrence $s = 2p/(n-1)$ is generated. If $r = 1$, the arc $X_i - X_j$ is added to $\mathcal{G}$.
  3. A random topological ordering π of the variables is generated.
  4. The UG $\mathcal{G}$ is transformed into a DAG by directing the arcs in such way that the resulting graph is consistent with π.
- Generation of the parameters
  1. For each variable, its number of states is generated using a uniform distribution between 2 and m.
  2. For each variable X_i, each configuration of its parents j and each state of the variable k, a random set of values r_{ijk} is generated uniformly distributed between 0 and 1.
  3. The CPT for each variable X_i is generated via:

$$p(X_i = k | Pa(X_i) = j) = \frac{{r_{ijk}}^2}{\sum_k {r_{ijk}}^2}.$$

The following parameters have been chosen to generate the networks: 50, 100 and 200 variables; 1 and 2 as the connectivity factor of the network; and 2 and 5 as maximum number of states of the variables. For each combination of the parameters, we generated 30 random networks; and for each network, a dataset containing 5000 instances was sampled. The characteristics of the generated networks are shown in Table 9. Each row represents the average value over the 30 datasets for a specific combination of the parameters, and the values in brackets represent the minimum–maximum values for these sets of networks. A total of 360 Bayesian networks were generated and tested.

Figs. 4–6 show the results obtained by the algorithms. Detailed results are shown in Tables A.13, A.15 and A.17 (Appendix A). In the graphs, the x-axis represents the execution time of the algorithms in seconds, the y-axis represents the BDeu score, and the radius of the circles is proportional to the logarithmic of the maximum CPT size. A look at the graphs gives us an idea of the overall performance of the algorithms. These graphs can be observed with a non-dominance criterion with respect to the time and the score, so an algorithm is dominated by another one if it is worse both in time and score.

An analysis of the main patterns that we extract from the graphs is given below.

- By using the non-dominance criterion it be can seen in the graphs that, in general, the non-dominated algorithms are: GES, GES_{G*}, GES_{iC} and GES_{iCG}, listed in order of best BDeu score.

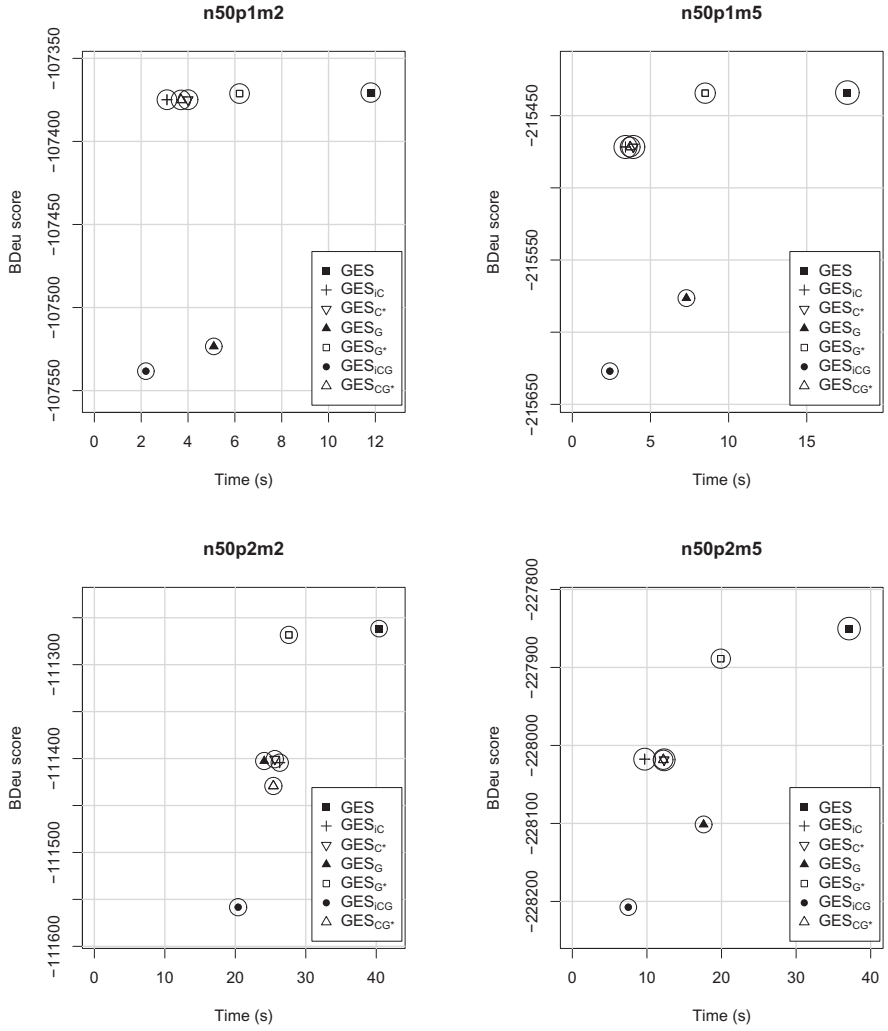

Fig. 4. Results for $n = 50$.

- By using the same criterion, but listed in Time-based order, we obtain the inverse order.
- GES always appears at the top right corner, i.e. GES obtains the best networks in many cases, but it is the slowest algorithm.
- GES$_{G*}$ obtains BDeu results that are close to the results obtained by GES, but it is almost two times faster.
- GES$_{iCG}$ is very fast. Although its results are not as good as the other algorithms, it is still close to GES, and saves a considerable amount of memory.
- GES$_{C*}$, GES$_{CG*}$ and GES$_{iC}$ obtain similar results, in some cases with BDeu scores close to those of GES and GES$_{G*}$. GES$_{iC}$ is, in general, the fastest one.
- GES$_{G}$ and GES$_{iCG}$ are the algorithms that create the smallest CPTs. This aspect is very important in environments where memory size may be limited and/or with very complex domains, as occurred in the HailFinder domain in the previous Section 7.2.1. We can observe this phenomenon in the graphs with the circle size surrounding the corresponding algorithm, especially with $n = 200$, 100 and $p = 1$.

Besides the performance indicators shown in the graphs, we also show a summary of the SHD results, the number of computations of the metric, and the maximum numbers of neighbors to compute an operation $X \rightarrow Y$, in Tables 10, 11 and 12 respectively. Detailed results are shown in Tables A.14, A.16 and A.18 (Appendix A). The behavior of the algorithms shown in these tables is similar to the behavior of the algorithms with real networks. The main conclusion from this set of experiments is that the differences in SHD are quite small. However, our algorithms perform much fewer computations of the metric. Finally, the differences in the maximum number of neighbors evaluated show that the algorithms that perform a greedy search of the subset **T** need to evaluate a smaller number of subsets.

We perform a statistical analysis of the results (a Wilcoxon test for pairwise comparisons [35]) in order to compare the results obtained by our algorithms versus the results obtained by the original GES algorithm in each set of networks.

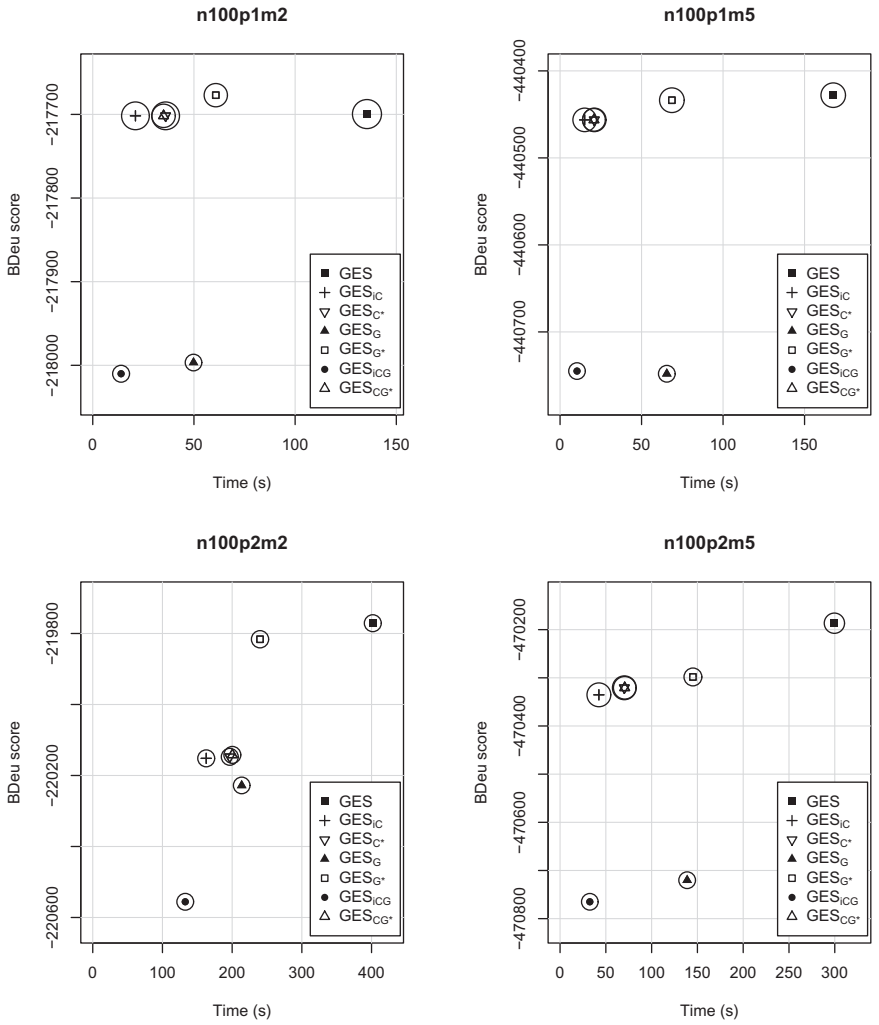

Fig. 5. Results for $n = 100$.

Table 10
Structural difference with the original network.

	GES	GES$_{iC}$	GES$_{C*}$	GES$_{G}$	GES$_{G}$*	GES$_{iCG}$	GES$_{CG*}$
n50	**20.5**	20.7	20.8	21.6	20.6	22.0	20.9
n100	45.3	45.6	45.5	46.0	**44.7**	47.2	45.6
n200	**95.4**∗	98.0∗	98.0∗	98.3	96.3	100.1	98.0

Table 11
Number of computations of the metric.

	GES	GES$_{iC}$	GES$_{C*}$	GES$_{G}$	GES$_{G}$*	GES$_{iCG}$	GES$_{CG*}$
n50	8 507.7	5 956.6	6 022.5	4 750.8	6 426.5	**4 176.8**	5 940.0
n100	38 067.6	24 383.2	24 635.1	18 918.7	26 544.5	**16 724.9**	24 378.4
n200	254 697.9∗	133 664.5∗	134 693.5∗	77 860.4	129 390.8	**68 645.1**	111 386.9

Table 12
Maximum numbers of neighbors visited to compute one insert operation.

	GES	GES$_{iC}$	GES$_{C*}$	GES$_{G}$	GES$_{G}$*	GES$_{iCG}$	GES$_{CG*}$
n50	15.0	15.3	15.3	**5.7**	12.4	**5.7**	12.5
n100	57.5	48.9	48.9	**6.2**	29.5	**6.2**	27.2
n200	646.6∗	2 796.9∗	2 796.9∗	8.3	182.7	**8.2**	131.6

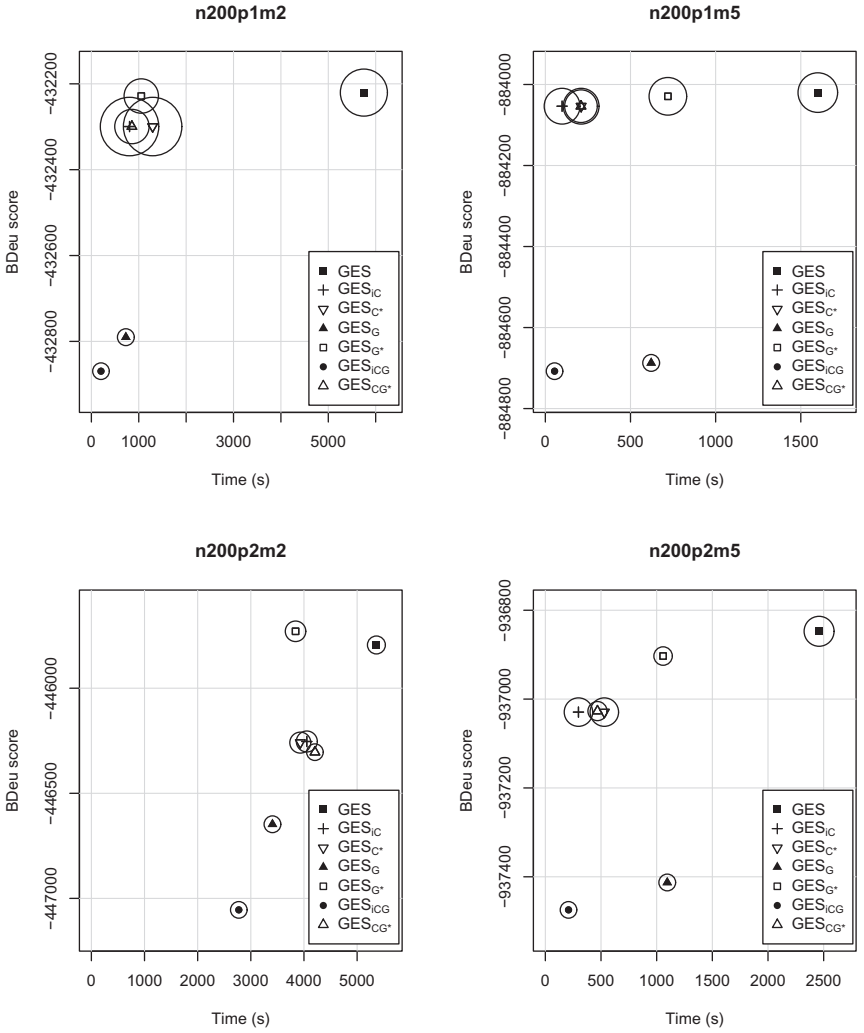

Fig. 6. Results for $n = 200$.

The confidence level was set to 95% in all the tests, which were performed for each one of the performance indicators. The results are shown in Appendix A. The most noteworthy conclusions that can be extracted from the statistical analysis of the performance indicators are the following:

- BDeu results: There is no statistical difference between GES and GES$_{G*}$ in 10 of the sets of networks. GES$_{iC}$, GES$_{C*}$ and GES$_{CC*}$ also obtain good BDeu results. In 4 cases there are no statistical differences between these algorithms and GES.
- SHD results: In general, there are no differences between GES and GES$_{G*}$, GES$_{iC}$, GES$_{C*}$ and GES$_{CC*}$. GES$_{iCG}$ also obtains good results, since in 4 cases there are no statistical differences with respect to GES.
- Learning time and number of computations: Our algorithms are faster and perform less computations of the metric than GES. The differences are significant in all the cases.
- CPT size and number of neighbors: The differences between GES and those algorithms that made the search of **T** greedily are statistically significant in the majority of the cases.

7.3.1. Behavior of the algorithms in relation with the number of instances

In this section we report the results obtained by the algorithms with different number of instances in the databases. We sampled 6 datasets, 5 for training and one for testing, from each one of the 360 synthetic networks. The number of instances of the training datasets is 1000, 2500, 5000, 10000 and 20000, and the number of instances of each test dataset is 20000.

For each combination of algorithm and synthetic networks, we ran the algorithm with each one of the five training databases and then we measured the execution time and the BDeu score of the output network over the test dataset.

Fig. 7 shows the average BDeu score over the test of the learned networks. Each point in the graph is the average result of the algorithm on the 360 synthetic networks. Fig. 7(a) displays the absolute values, whereas Fig. 7(b) represents the

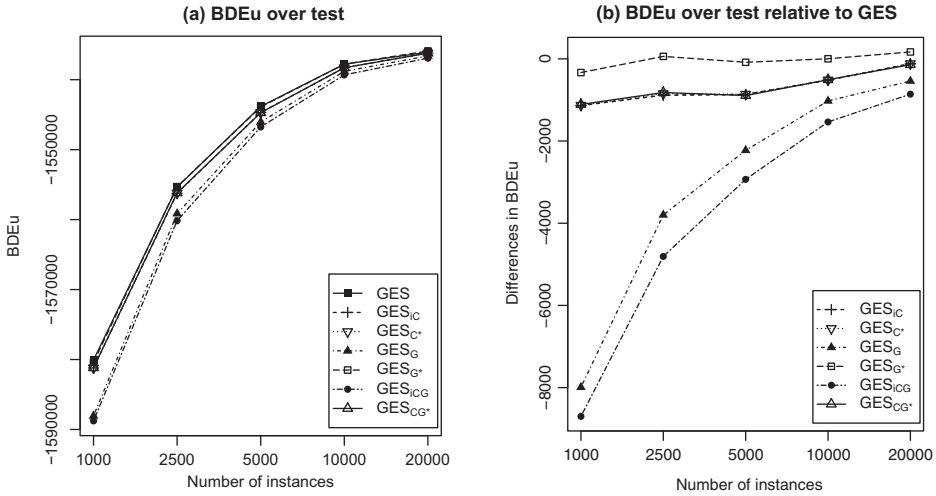

Fig. 7. BDeu score and differences with respect to GES.

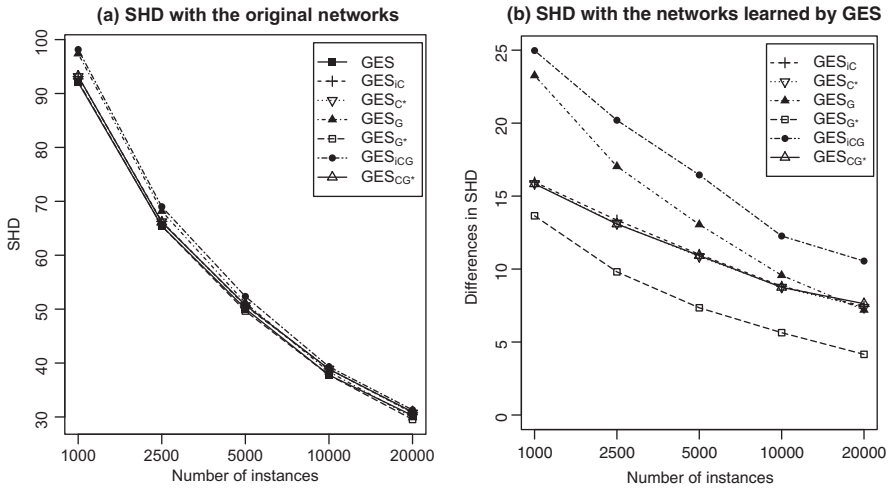

Fig. 8. SHD with respect to the original networks and with those learned by GES.

differences with respect to GES. The graphs shown that the quality of the networks increases and their relative differences decrease as the number of instances grows large.

Fig. 8(a) shows the SHD between the networks learned by the algorithms and the original ones. The graph shows that the SHD decreases as the number of instances increases. In Fig. 8(b), it is shown the SHD between the networks learned by the proposed algorithms and those learned by GES. In this case, the graph shows that differences with GES are small, and also decrease as the number of instances grows large.

Finally, Fig. 9 shows the execution time of the algorithms averaged over the set of synthetic networks. The graph shows that the execution time increases as the number of instances increases, but the relative differences between the algorithms are almost preserved.

8. Conclusions

In this paper we have presented several variants of the original GES algorithm. The first one, GES$_C$, and its iterative version, GES$_{iC}$, are able to reduce the search space by constraining it during its execution. The second one, GES$_G$, performs a greedy search in the space of candidate neighbors, avoiding the need to carry out an exhaustive search. The third group of algorithms, composed of GES$_{CG}$ and GES$_{iCG}$, combine the advantages of both approaches. Finally, the last group of algorithms comprised GES$_{G^*}$, GES$_{C^*}$ and GES$_{CG^*}$. This group of algorithms perform two forward iterations. In the first one, a greedy or a constrained forward phase is executed, whereas in the second phase the original FES algorithm is executed.

Moreover, apart from the definition of the algorithms, we have also proved that, under the faithfulness assumption, all of the iterative algorithms and GES$_G$ are asymptotically optimal.

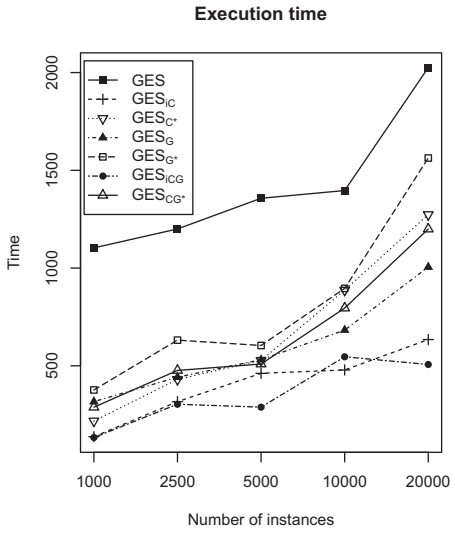

Fig. 9. Execution time of the algorithms.

The experimental evaluation of our algorithms has been carried out by measuring the accuracy and the efficiency of the algorithms. We have also tested the algorithms with different numbers of instances in the datasets. In terms of accuracy, the results obtained by GES_{G^*} are comparable to those obtained by GES in the majority of our test networks and regardless of the number of instances in the datasets. Also, the results of GES_{iC}, GES_{C^*} and GES_{CG^*} are quite good in general. Finally, it is important to remark that the differences between the asymptotically optimal algorithms are very small when the number of instances is sufficiently large. Regarding efficiency, our algorithms are faster and perform less computations of the score metric than GES. Moreover, we have also analyzed the efficiency of the algorithms in terms of memory requirements, and we have found that the algorithms that perform a forward greedy search require less memory than the other algorithms. Finally, we should also highlight the fact that algorithms performing a forward greedy search are able to work with networks that the other algorithms are unable to.

In the future we plan to use the proposed variants of the GES algorithm as the core of other metaheuristic techniques based on the space of equivalence classes. In this way, we will try to improve the accuracy of the solutions found by GES while still maintaining the controlled runtime of the algorithms.

Acknowledgements

This work has been partially funded by FEDER funds, the Spanish Government (MICINN), and the Regional Government of Castilla-La Mancha through projects TIN2010-20900-C04-03 and PCI08-0048-8577. Juan I. Alonso is also funded by a JCCM research grant associated to the project with reference number PCI08-0048.

Appendix A. Results

This appendix shows detailed results of the experiments carried out in Section 7.3. The results of the GES algorithm are absolute results presented in their corresponding measure. The results of our proposed algorithms are presented relative to the results of GES. For the BDeu score and the SHD the results in the tables are the difference between the results obtained by the algorithms and the results obtained by GES. For the other performance indicators, the results in the tables represent the results obtained by the algorithms divided by the results obtained by GES.

Each cell in the tables represents the average value obtained by running the algorithms over a set of 30 randomly generated networks. Bold numbers indicate the best results for the corresponding set of networks and underlined numbers indicate that the results do not present statistical differences with the results obtained by GES. In practice, networks within a set of networks are very different and it is possible to have uneven results within them. Beside each average result we report the standard deviation (in brackets in the tables). The standard deviation in the column of GES should be interpreted as a measure of variability within each set of networks. The standard deviation of the other algorithms represents the variability of the differences between the algorithms and GES, as these results are relative to GES. Note that the relative results presented are the average of the relative individual results; i.e. for each individual execution we subtract or divide by the result of the execution of GES for the same dataset and then we take the mean value and the standard deviation.

It is important to mention the fact that GES, GES_C and GES_{iC} where unable to recover two networks of the set labeled *n200p1m2* due to memory limitations. The cause of this error is identical to the problem with the HailFinder network, which

Table A.13
BDeu score of the networks.***

	GES		Differences with respect to GES											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
n50p1m2	**−107 370.6**	(8 009.2)	−4.2	(14.6)	−4.2	(14.6)	−152.8	(299.0)	−0.6	(2.3)	−167.6	(297.4)	−4.3	(14.6)
n50p1m5	**−215 434.0**	(13 590.1)	−37.6	(182.8)	−37.6	(182.8)	−142.4	(274.2)	−0.4	(2.3)	−193.0	(302.5)	−37.6	(182.8)
n50p2m2	**−111 261.6**	(5 898.2)	−142.8	(383.1)	−139.1	(381.7)	−141.0	(401.4)	−6.8	(276.0)	−296.7	(485.1)	−167.6	(396.8)
n50p2m5	**−227 850.2**	(13 850.3)	−167.4	(403.4)	−168.2	(404.8)	−251.1	(350.9)	−38.6	(168.9)	−357.2	(550.1)	−168.2	(404.8)
n100p1m2	−217 699.6	(11 221.7)	−2.1	(183.6)	−2.1	(183.6)	−297.1	(383.7)	**22.6**	(119.5)	−310.6	(386.8)	−2.1	(183.6)
n100p1m5	**−440 427.9**	(19 928.3)	−28.4	(144.9)	−28.4	(144.9)	−320.2	(532.5)	−5.7	(58.4)	−317.1	(527.5)	−28.4	(144.9)
n100p2m2	**−219 771.0**	(9 696.5)	−380.7	(553.9)	−376.4	(563.5)	−457.0	(542.8)	−45.2	(235.4)	−784.8	(711.2)	−371.0	(560.8)
n100p2m5	**−470 186.6**	(22 029.2)	−148.7	(359.7)	−134.0	(326.6)	−533.6	(658.3)	−111.6	(248.1)	−578.4	(790.2)	−134.0	(326.6)
n200p1m2	−432 272.6*	(15 752.0)	−69.5*	(203.7)	−69.5*	(203.7)	−560.2	(547.7)	**1.6**	(40.7)	−639.5	(572.9)	−69.5	(203.7)
n200p1m5	**−884 020.0**	(18 546.4)	−33.7	(87.7)	−33.7	(87.7)	−667.4	(844.5)	−9.3	(27.2)	−687.7	(846.3)	−33.7	(87.7)
n200p2m2	−445 794.3	(13 291.6)	−459.1	(1 210.8)	−464.6	(1 208.2)	−853.0	(1 506.5)	**65.1**	(1 140.6)	−1 260.3	(1 433.5)	−510.0	(1 220.1)
n200p2m5	**−936 847.4**	(21 675.5)	−181.9	(440.6)	−181.9	(440.6)	−565.4	(696.3)	−55.7	(387.8)	−627.1	(680.1)	−179.6	(441.4)

Table A.14
Structural difference with the original network.

	GES		Differences with respect to GES											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
50p1m2	**17.4**	(8.9)	0.6	(2.8)	0.6	(2.8)	0.8	(4.5)	0.5	(2.8)	0.9	(4.5)	0.6	(2.8)
50p1m5	7.1	(5.9)	**−0.1**	(2.1)	**−0.1**	(2.1)	1.0	(2.2)	0.1	(0.5)	1.1	(2.0)	**−0.1**	(2.1)
50p2m2	25.0	(12.9)	−0.4	(8.7)	−0.1	(8.7)	0.9	(6.1)	**−0.6**	(5.8)	1.9	(9.0)	0.5	(8.9)
50p2m5	**32.5**	(9.5)	0.7	(3.4)	0.7	(3.4)	1.9	(3.7)	0.3	(3.5)	2.2	(4.8)	0.7	(3.4)
100p1m2	47.2	(9.1)	0.3	(4.8)	0.3	(4.8)	**−2.1**	(5.3)	−0.5	(4.6)	−1.2	(6.3)	0.3	(4.8)
100p1m5	13.3	(6.3)	0.1	(2.3)	0.1	(2.3)	1.6	(3.0)	**−0.1**	(1.4)	1.9	(2.8)	0.1	(2.3)
100p2m2	52.0	(16.1)	−0.2	(11.8)	−0.5	(11.9)	−0.7	(9.3)	**−2.8**	(9.3)	3.1	(12.7)	−0.03	(11.9)
100p2m5	**68.7**	(11.8)	1.1	(3.5)	1.0	(3.2)	4.0	(4.3)	1.0	(3.0)	3.8	(4.0)	1.0	(3.2)
200p1m2	113.9*	(20.0)	1.3*	(6.7)	1.3*	(6.7)	**−0.8**	(7.9)	0.8	(7.5)	−0.2	(7.8)	1.3	(6.7)
200p1m5	**31.8**	(12.6)	1.7	(3.5)	1.7	(3.5)	4.0	(4.2)	0.9	(2.9)	4.9	(4.5)	1.7	(3.5)
200p2m2	**101.2**	(18.3)	4.4	(16.5)	4.5	(16.3)	2.7	(16.0)	0.03	(16.6)	7.6	(17.0)	4.8	(16.5)
200p2m5	**134.8**	(23.2)	2.7	(3.4)	2.7	(3.4)	5.5	(4.7)	1.7	(3.1)	6.4	(5.6)	2.6	(3.5)

Table A.15
Learning time.

	GES		Relative values (divided by GES results)											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
50p1m2	11.8	(4.7)	0.26	(0.12)	0.35	(0.08)	0.47	(0.15)	0.56	(0.12)	**0.19**	(0.05)	0.32	(0.08)
50p1m5	17.6	(6.1)	0.20	(0.07)	0.23	(0.06)	0.44	(0.12)	0.51	(0.12)	**0.14**	(0.04)	0.22	(0.05)
50p2m2	40.4	(16.2)	0.61	(0.46)	0.64	(0.29)	0.60	(0.19)	0.69	(0.17)	**0.49**	(0.35)	0.63	(0.25)
50p2m5	37.1	(13.1)	0.26	(0.08)	0.33	(0.08)	0.50	(0.17)	0.56	(0.12)	**0.21**	(0.06)	0.33	(0.07)
100p1m2	135.5	(40.7)	0.15	(0.06)	0.25	(0.14)	0.38	(0.08)	0.46	(0.10)	**0.10**	(0.03)	0.25	(0.13)
100p1m5	167.6	(46.7)	0.09	(0.03)	0.13	(0.04)	0.41	(0.09)	0.43	(0.08)	**0.06**	(0.02)	0.13	(0.04)
100p2m2	401.8	(187.1)	0.40	(0.22)	0.50	(0.22)	0.56	(0.19)	0.63	(0.21)	**0.32**	(0.21)	0.50	(0.25)
100p2m5	299.6	(67.0)	0.14	(0.04)	0.24	(0.06)	0.47	(0.09)	0.49	(0.10)	**0.11**	(0.03)	0.24	(0.05)
200p1m2	5 751.2*	(18 698.8)	0.36*	(0.72)	0.48*	(0.77)	0.41	(0.24)	0.46	(0.22)	**0.11**	(0.11)	0.26	(0.28)
200p1m5	1 599.9	(504.7)	0.06	(0.04)	0.13	(0.10)	0.41	(0.07)	0.45	(0.10)	**0.03**	(0.01)	0.13	(0.10)
200p2m2	5 363.1	(2 332.8)	0.75	(1.44)	0.74	(1.09)	0.64	(0.29)	0.71	(0.41)	**0.52**	(0.75)	0.80	(1.32)
200p2m5	2 458.9	(526.0)	0.12	(0.06)	0.22	(0.08)	0.45	(0.13)	0.44	(0.11)	**0.09**	(0.05)	0.19	(0.07)

Table A.16
Number of computations of the metric.

	GES		Relative values (divided by GES results)											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
50p1m2	8 089.9	(2 665.5)	0.74	(0.12)	0.75	(0.12)	0.53	(0.10)	0.76	(0.12)	**0.49**	(0.09)	0.74	(0.13)
50p1m5	7 092.9	(1 120.4)	0.76	(0.05)	0.76	(0.05)	0.52	(0.07)	0.80	(0.05)	**0.49**	(0.06)	0.76	(0.05)
50p2m2	9 979.9	(738.2)	0.67	(0.09)	0.69	(0.09)	0.63	(0.05)	0.75	(0.07)	**0.52**	(0.04)	0.67	(0.08)
50p2m5	8 868.3	(1 360.4)	0.65	(0.07)	0.66	(0.07)	0.58	(0.06)	0.72	(0.06)	**0.49**	(0.07)	0.65	(0.07)
100p1m2	45 111.5	(2 6694.9)	0.65	(0.18)	0.65	(0.18)	0.45	(0.15)	0.67	(0.20)	**0.43**	(0.14)	0.64	(0.19)
100p1m5	29 265.8	(2 716.1)	0.74	(0.07)	0.74	(0.07)	0.50	(0.04)	0.77	(0.07)	**0.47**	(0.05)	0.74	(0.07)
100p2m2	42 846.9	(4 497.4)	0.60	(0.06)	0.62	(0.06)	0.58	(0.06)	0.73	(0.07)	**0.48**	(0.04)	0.62	(0.07)
100p2m5	35 046.3	(2 526.3)	0.65	(0.05)	0.66	(0.05)	0.55	(0.05)	0.70	(0.04)	**0.48**	(0.04)	0.65	(0.05)
200p1m2	530 219.3*	(1 080 148.2)	0.89*	(1.09)	0.90*	(1.10)	0.37	(0.22)	0.53	(0.35)	**0.35**	(0.20)	0.51	(0.35)
200p1m5	167 441.3	(14 3261.9)	0.72	(0.14)	0.73	(0.14)	0.43	(0.12)	0.78	(0.07)	**0.40**	(0.11)	0.72	(0.14)
200p2m2	176 556.6	(18 410.9)	0.55	(0.05)	0.57	(0.05)	0.58	(0.05)	0.73	(0.41)	**0.47**	(0.04)	0.56	(0.05)
200p2m5	144 574.3	(9 023.1)	0.61	(0.03)	0.62	(0.03)	0.55	(0.04)	0.68	(0.03)	**0.47**	(0.03)	0.62	(0.03)

Table A.17
Size of the biggest CPT.

	GES		Relative values (divided by GES results)											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
50p1m2	121.6	(177.0)	1.22	(1.28)	1.22	(1.28)	**0.90**	(0.26)	1.19	(1.30)	0.94	(0.33)	1.19	(1.30)
50p1m5	3 102.0	(6 608.5)	0.92	(0.30)	0.92	(0.30)	0.72	(0.30)	0.95	(0.58)	**0.71**	(0.33)	0.88	(0.34)
50p2m2	**236.8**	(139.8)	1.22	(0.45)	1.22	(0.45)	1.13	(0.35)	1.13	(0.35)	1.18	(0.43)	1.18	(0.43)
50p2m5	3 695.2	(4 463.4)	0.93	(0.24)	0.93	(0.24)	0.75	(0.32)	0.83	(0.29)	**0.73**	(0.33)	0.82	(0.30)
100p1m2	700.8	(1 228.1)	1.12	(0.44)	1.12	(0.44)	**0.69**	(0.42)	0.80	(0.37)	0.78	(0.58)	0.87	(0.53)
100p1m5	4 288.3	(8 369.4)	1.13	(1.38)	1.13	(1.38)	0.65	(0.31)	1.03	(1.40)	**0.64**	(0.31)	1.01	(1.41)
100p2m2	384.0	(130.2)	1.05	(0.36)	1.05	(0.36)	**0.98**	(0.25)	1.07	(0.25)	1.03	(0.37)	1.05	(0.36)
100p2m5	3 153.7	(1 845.8)	1.43	(2.03)	1.43	(2.03)	**0.74**	(0.29)	0.83	(0.35)	**0.74**	(0.29)	1.30	(2.06)
200p1m2	8 940.1*	(24 568.8)	110.45*	(418.09)	110.45*	(418.09)	0.35	(0.41)	0.57	(0.79)	**0.34**	(0.41)	0.56	(0.79)
200p1m5	40 969.7	(99 948.0)	0.92	(0.24)	0.92	(0.24)	0.47	(0.40)	0.85	(0.32)	**0.46**	(0.39)	0.81	(0.36)
200p2m2	452.3	(347.3)	2.03	(5.67)	2.03	(5.67)	**0.98**	(0.27)	2.01	(5.67)	**0.98**	(0.35)	**0.98**	(0.35)
200p2m5	13 263.5	(27 011.7)	0.85	(0.20)	0.85	(0.20)	0.45	(0.32)	0.51	(0.31)	**0.44**	(0.31)	0.52	(0.32)

Table A.18
Maximum numbers of neighbors visited to compute one insert operation.

	GES		Relative values (divided by GES results)											
			GES_{iC}		GES_{C*}		GES_{G}		$GES_{G}*$		GES_{iCG}		GES_{CG*}	
50p1m2	23.1	(44.8)	1.27	(1.28)	1.27	(1.28)	**0.45**	(0.21)	0.96	(1.36)	**0.45**	(0.21)	0.96	(1.36)
50p1m5	12.4	(8.4)	0.98	(0.25)	0.98	(0.25)	**0.53**	(0.22)	0.88	(0.19)	**0.52**	(0.23)	0.89	(0.29)
50p2m2	10.9	(3.9)	1.13	(0.35)	1.13	(0.35)	**0.66**	(0.19)	0.90	(0.29)	**0.66**	(0.19)	0.94	(0.34)
50p2m5	13.7	(9.0)	0.90	(0.20)	0.90	(0.20)	0.50	(0.22)	0.74	(0.27)	**0.49**	(0.22)	0.73	(0.27)
100p1m2	167.5	(310.7)	1.17	(0.68)	1.17	(0.68)	0.25	(0.23)	0.71	(0.82)	**0.25**	(0.23)	0.69	(0.82)
100p1m5	16.5	(11.9)	0.97	(0.13)	0.97	(0.13)	**0.41**	(0.19)	0.88	(0.22)	**0.41**	(0.19)	0.87	(0.22)
100p2m2	32.0	(36.6)	1.09	(0.47)	1.09	(0.47)	**0.42**	(0.25)	0.87	(0.40)	0.43	(0.25)	0.72	(0.39)
100p2m5	14.1	(5.3)	1.12	(0.58)	1.12	(0.58)	**0.49**	(0.21)	0.78	(0.27)	**0.49**	(0.21)	0.91	(0.64)
200p1m2	2 230.1*	(6 144.1)	181.11*	(776.43)	181.11*	(776.43)	**0.19**	(0.30)	0.50	(0.84)	**0.19**	(0.30)	0.50	(0.84)
200p1m5	276.8	(770.8)	0.97	(0.17)	0.97	(0.17)	**0.26**	(0.23)	0.86	(0.24)	**0.26**	(0.23)	0.83	(0.28)
200p2m2	51.5	(97.8)	3.20	(11.49)	3.20	(11.49)	0.34	(0.16)	2.69	(11.59)	**0.33**	(0.16)	0.55	(0.49)
200p2m5	28.0	(24.5)	1.02	(0.21)	1.02	(0.21)	**0.36**	(0.18)	0.62	(0.38)	**0.36**	(0.18)	0.65	(0.37)

is detailed in Section 7.2.1. We have highlighted these executions with an asterisk to highlight the fact that the average results in the tables are influenced by these errors.

References

[1] R. Neapolitan, Learning Bayesian Networks, Prentice Hall, 2003.
[2] N. García-Pedrajas, A. de Haro-García, Scaling up data mining algorithms: review and taxonomy, Progress in Artificial Intelligence 1 (1) (2012) 71–87.
[3] D.M. Chickering, Learning Bayesian networks is NP-complete, in: D. Fisher, H. Lenz (Eds.), Learning from Data: Artificial Intelligence and Statistics V, Springer-Verlag, 1996, pp. 121–130.
[4] P. Parviainen, M. Koivisto, Exact structure discovery in Bayesian networks with less space, in: Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence (UAI'09), 2009, pp. 436–443.
[5] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Machine Learning 20 (3) (1995) 197–243.
[6] D.M. Chickering, Optimal structure identification with greedy search, Journal of Machine Learning Research 3 (2002) 507–554.
[7] N. Friedman, I. Nachman, D. Pe'er, Learning Bayesian network structure from massive datasets: the Sparse Candidate Algorithm, in: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI'99) (1999) 206–215.
[8] D.M. Chickering, D. Geiger, D. Heckerman, Learning Bayesian networks: search methods and experimental results, in: Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics (1995) 112–128.
[9] I. Tsamardinos, L.E. Brown, C.F. Aliferis, The max–min hill-climbing Bayesian network structure learning algorithm, Machine Learning 65 (1) (2006) 31–78.
[10] P. Larrañaga, M. Poza, Y. Yurramendi, R.H. Murga, C.M.H. Kuijpers, Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (9) (1996) 912–926.
[11] S. Acid, L.M. de Campos, Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs, Journal of Artificial Intelligence Research 18 (2003) 445–490.
[12] L.M. de Campos, J.M. Fernández-Luna, J.A. Gámez, J.M. Puerta, Ant colony optimization for learning Bayesian networks, International Journal of Approximate Reasoning 31 (3) (2002) 291–311.
[13] R. Blanco, I. Inza, P. Larrañaga, Learning Bayesian networks in the space of structures by estimation of distribution algorithms, International Journal of Intelligent Systems 18 (2) (2003) 205–220.
[14] W. Lam, F. Bacchus, Learning bayesian belief networks: an approach based on MDL principle, Computational Intelligence 10 (1994) 269–293.
[15] G. Schwartz, Estimating the dimension of a model, Annals of Statistics 6 (1978) 461–464.
[16] L.M. de Campos, A scoring function for learning Bayesian networks based on mutual information and conditional independence tests, Journal of Machine Learning Research 7 (2006) 2149–2187.
[17] P. Spirtes, C. Glymour, R. Scheines, Causation, Prediction and Search, Lecture Notes in Statistics, vol. 81, Springer-Verlag, 1993.
[18] D. Dash, M.J. Druzdzel, 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), 1999, pp. 142–149.
[19] J.D. Nielsen, T. Kocka, J. Peña, On local optima in learning Bayesian networks, in: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03), 2003, pp. 435–442.

[20] J.I. Alonso-Barba, L. delaOssa, J.M. Puerta, J.A. Gamez, Scaling up the Greedy Equivalence Search algorithm by constraining the search space of equivalence classes, in: Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-2011), 2011, pp. 194–205.
[21] J.A. Gámez, J.L. Mateo, J.M. Puerta, Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood, Data Mining and Knowledge Discovery 22 (1–2) (2011) 106–148.
[22] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[23] F. Jensen, T. Nielsen, Bayesian Networks and Decision Graphs, Springer, 2007.
[24] T. Verma, J. Pearl, Equivalence and synthesis of causal models, in: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI'90), Elsevier Science Inc., 1991, pp. 255–270.
[25] M. Studený, Jirí Vomlel, Raymond Hemmecke, A geometric view on learning Bayesian network structures, International Journal of Approximate Reasoning 51 (5) (2010) 573–586.
[26] R. Hemmecke, S. Lindner, M. Studený, Characteristic imsets for learning Bayesian network structure, International Journal of Approximate Reasoning 53(9) (2012) 1336–1349 .
[27] M. Studený, Jirí Vomlel, On open questions in the geometric approach to structural learning Bayesian nets, International Journal of Approximate Reasoning 52 (5) (2011) 627–640.
[28] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, G.F. Cooper, The alarm monitoring system: a case study with two probabilistic inference techniques for belief networks, in: Second European Conference on Artificial Intelligence in Medicine, vol. 38, Springer-Verlag, Berlin, 1989, pp. 247–256.
[29] K. Kristensen, I.A. Rasmussen, 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 (2002) 197–217.
[30] B. Abramson, J. Brown, W. Edwards, A. Murphy, R.L. Winkler, Hailfinder: a Bayesian system for forecasting severe weather, International Journal of Forecasting 12 (1) (1996) 57–71.
[31] J. Binder, D. Koller, S.J. Russell, K. Kanazawa, Adaptive probabilistic networks with hidden variables, Machine Learning 29 (2–3) (1997) 213–244.
[32] A. Jensen, F. Jensen, 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) (1996) 349–356.
[33] S. Andreassen, F.V. Jensen, S.K. Andersen, B. Falck, U. Kjrulff, M. Woldbye, A.R. Srensen, A. Rosenfalck, F. Jensen, MUNIN – an expert EMG assistant, in Computer-Aided Electromyography and Expert Systems, John E. Desmedt (ed.) Elsevier Science Publishers, (1989) Chapter 21.
[34] C.S. Jensen, Blocking Gibbs sampling for inference in large and complex Bayesian networks with applications in genetics, Ph.D. thesis, Aalborg University, Denmark, 1997.
[35] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.