Soft Comput (2011) 15:1881–1895
DOI 10.1007/s00500-010-0623-x

FOCUS

Structural learning of Bayesian networks using local algorithms based on the space of orderings

Juan I. Alonso-Barba · Luis delaOssa · Jose M. Puerta

Published online: 16 June 2010

Abstract Structural learning of Bayesian networks (BNs) is an NP-hard problem which is generally addressed by means of heuristic search algorithms. Despite the fact that earlier proposals for dealing with this task were based on searching the space of Directed Acyclic Graphs (DAGs), there are some alternative approaches. One of these approaches for structural learning consists of searching the space of orderings, as given a certain topological order among the problem variables, it is relatively easy to build (and evaluate) a BN compatible with it. In practice, the latter methods make it possible to obtain good results, but they are still costly in terms of computation. In this article, we prove the correctness of the method used to evaluate each ordering, and we propose some efficient learning algorithms based on it. Our first proposal is based on the Hill-Climbing algorithm, and uses an improved neighbourhood definition. The second algorithm is an extension of the first one, and is based on the well-known Variable Neighbourhood Search metaheuristic. Finally, iterative versions of both algorithms are also proposed. The algorithms have been tested over a set of different domains, and have been compared with other methods such as Hill-Climbing in the space of DAGs or *Greedy Equivalent Search*, in order to study their behaviour in practice.

Keywords Bayesian networks · Structural learning · Orderings · Variable Neighbourhood Search

Research Projects PCI08-0048-8577 and TIN2007-67418-C03-01.

J. I. Alonso-Barba (✉) · L. delaOssa · J. M. Puerta
Laboratory of Intelligent Systems and Data Mining,
Albacete Research Institute of Informatics,
University of Castilla-La Mancha,
02071 Albacete, Spain
e-mail: jialonso@dsi.uclm.es

L. delaOssa
e-mail: ldelaossa@dsi.uclm.es

J. M. Puerta
e-mail: jpuerta@dsi.uclm.es

1 Introduction

Learning Bayesian networks is a complex task which has received a lot of attention from researchers in the area. Since finding the optimal structure of a network given the data has been stated as an *NP-hard* problem (Chickering 1996), it becomes necessary to use heuristic and metaheuristic techniques to find quality solutions.

There are several approaches to address structural learning. Thus, in many related works the search for the network structure is carried out in the space of Directed Acyclic Graphs (DAGs). In this respect, there are several proposals based on greedy (Buntine 1991), local search (Heckerman et al. 1995), and population-based algorithms (Larrañaga et al. 1996; Blanco et al. 2003).

The Hill-Climbing (HC) algorithm on the space of DAGs is one of the most popular algorithms, since it presents a good trade-off between the quality of the network obtained and computational demands. Moreover, this algorithm guarantees that, assuming certain conditions, the network obtained by it is a minimal I-map of the target distribution. Some recent studies have tried to improve this algorithm by constraining the search in order to obtain more scalable algorithms. The Constrained Hill-Climbing algorithm (CHC) and the iterated CHC algorithm (iCHC) (Gámez et al. 2010) are based on the restriction of the candidate solutions to be evaluated during the search process when an independency between variables is found.

Another proposal in the same line, MaxMin Hill Climbing (MMHC) (Tsamardinos et al. 2006), is a two-step algorithm that in its first stage tries to identify the parents and children of each variable and in the second uses a local search algorithm to look for the network, but with the search restricted to the set of previously found adjacencies (parents and children). These algorithms achieve a significative reduction in computational demand with respect to the standard HC algorithm and are scalable to higher dimensionality domains. However, in Gámez et al. (2010), it is proved that these algorithms are statistically worse than the HC algorithm in terms of accuracy.

A different approach, which generally improves on the results obtained by the above methods, consists of searching the space of equivalence classes. One of the algorithms based on this, the *Greedy Equivalence Search* (GES) (Chickering 2002), is nowadays the algorithm of reference in Bayesian network learning. Under certain conditions, the final solution found by GES is guaranteed to be a *perfect-map* of the target distribution. Therefore, this method is considered to be asymptotically correct. However, its use is limited in the literature due to the fact that its implementation is complex. In fact, HC is included in almost all the BN software (such as `Weka` (Witten and Frank 2005), `Hugin`[1] or `SamIam`[2]) because of its easy implementation and its relatively good results in practice, whereas GES is hardly ever included. One of the very few reliable and open implementations of GES is the one included in `Tetrad`[3].

An alternative to the aforementioned approaches consists of searching the space of orderings among the domain variables. Thus, given a certain topological order, it is possible to obtain a compatible Bayesian network (Buntine 1991; de Campos and Puerta 2001). Based on this, some methods assign to each order the score of the network built from it, and then use search algorithms which traverse the permutation space.

One of the main advantages of this approach is that each step in the search makes a more global modification of the current state, thereby avoiding local maxima more effectively. Moreover, there are three more advantages related with efficiency: (1) the dimension of the search space is reduced; (2) the need to perform acyclicity checks on candidate successors is avoided; (3) it is easy to implement efficient ordering-based algorithms. Therefore, ordering-based methods make it possible to obtain good results by efficiently exploring the search space. However, they are very costly in terms of computation, since a different Bayesian network must be built on each evaluation.

[1] http://www.hugin.com/.

[2] http://reasoning.cs.ucla.edu/samiam/.

[3] http://www.phil.cmu.edu/projects/tetrad/.

In this study, we first propose an efficient method to evaluate each permutation, and we demonstrate that, asymptotically, it returns the minimal consistent *I-map* given the ordering. Afterwards, we propose several efficient methods which explore the space of orderings by means of *Hill Climbing* and *Variable Neighbourhood Search* (VNS) algorithms (Mladenović and Hansen 1997). These methods are based on the use of a particular neighbourhood operator which enables a significant reduction in the calculations. Nevertheless, experiments show that the results obtained with these techniques do not present a statistical difference with respect to those obtained with *Greedy Equivalence Search* (GES), and improve on Hill Climbing defined over the space of DAGs.

This paper is structured into four sections besides this introduction. Section 2 introduces some aspects of Bayesian network learning through searching the space of orderings. Then, in Sect. 3 both the proposed algorithm and the improvements used to increase its efficiency are explained in detail. Afterwards an experimental evaluation is carried out in Sect. 4. Finally, Sect. 5 outlines some conclusions and sets out some lines for future work.

2 Learning Bayesian Networks: space of orderings

Bayesian Networks (BN) are graphical models that can efficiently represent n-dimensional probability distributions. This representation has two components that respectively codify qualitative and quantitative knowledge:

- A graphical structure, or more precisely, a DAG, $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, where the nodes in $\mathbf{V} = \{X_1, X_2, \ldots, X_n\}$ represent the random variables[4] from the problem being modelled and the topology of the graph, i.e., the arcs in $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$, encodes conditional (in)dependence relationships among the variables (by means of the presence or absence of direct connections between pairs of variables).
- A set of numerical parameters (Θ), usually conditional probability distributions drawn from the graph structure: For each variable $X_i \in \mathbf{V}$ there is a conditional probability distribution $P(X_i | pa_{\mathcal{G}}(X_i))$, where $pa_{\mathcal{G}}(X_i)$ represents any combination of the values of the variables in $Pa_{\mathcal{G}}(X_i)$, and $Pa_{\mathcal{G}}(X_i)$ is the parent set of X_i in $\mathcal{G}$. From these conditional distributions we can recover the joint probability distribution over $\mathbf{V}$ because of the Markov Condition:

[4] 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).

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

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

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 affirmation, but related to a probability distribution p, is denoted by $I_p(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$. From these conditions, two definitions arise:

Definition 1 A DAG $\mathcal{G}$ is an *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 it is *minimal* if there is no arc such that, when removed, the resulting graph $\mathcal{G}'$ is still an *I-map*. $\mathcal{G}$ is a *D-map* of p if $\langle \mathbf{X}, \mathbf{Y} | \mathbf{Z} \rangle_{\mathcal{G}} \Leftarrow I_p(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$.

Definition 2 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). 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* (faithful) model (Pearl 1988). In the rest of this article, faithfulness will be assumed. Therefore, the terms *d-separation* and *conditional independence* will be used interchangeably.

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 $\mathbf{V}$, find a DAG $\mathcal{G}^*$ such that

$$\mathcal{G}^* = \arg\max_{\mathcal{G} \in \mathcal{G}^n} f(\mathcal{G} : D), \tag{1}$$

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.

There are many methods for finding $\mathcal{G}^*$. The most basic, based on Local Search algorithms (specifically Hill Climbing), traverse the search space of DAGs by starting from an initial solution and performing a finite number of steps. At each step, the algorithm only considers local changes, i.e. neighbour DAGs which only differ in one edge, and chooses the one which leads to the greatest improvement in f. Finally, the algorithm stops when there is no local change which improves f. Due to their nature, local algorithms often get trapped at local optima. In order to deal with this problem, different strategies such as restarts, randomness, etc., are used.

The effectiveness and efficiency of a local search procedure depends on several factors, such as the neighbourhood structure considered, the starting solution, or the capacity for fast evaluation of candidate subgraphs (neighbours). Thus, the neighbourhood structure is directly related to the operations used to generate neighbours. The usual choices to induce local changes when working in the space of DAGs are arc addition, arc deletion, and arc reversal. Thus, given a graph $\mathcal{G}$, there are $\mathcal{O}(n^2)$ possible changes, n being the number of variables. In the case of BN learning, this number is slightly smaller, since the application of the operators cannot produce any directed cycle.

In relation to the starting solution, there are several alternatives. Despite the fact that using an empty network is a valid option, random starting points or perturbed local optima are frequently used, specially in the case of iterated local search algorithms.

Finally, the efficient evaluation of neighbours of DAGs in Local Search algorithms, such as Hill-Climbing, 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:

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

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

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

If a decomposable metric is used, graphs resulting from 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. This fact makes it possible to reduce the complexity of each iteration from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

2.1 Search space of orderings

Early algorithms for BN structure learning, like K2 (Cooper and Herskovits 1992), assumed that a given ordering was known and then they searched for a network consistent with such an ordering. However, coming up with a good ordering requires a significant amount of domain knowledge, which is not commonly available in many practical applications. However, as these methods allow the transformation of an ordering into a BN, they provide a mechanism to evaluate such an ordering: evaluating THE network in question. Therefore, structural learning can be carried out by searching the space of orderings.

Given an ordering $\prec$, the possible parents sets for any given variable X_i can be defined as: $\mathcal{U}_{i,\prec} = \{\mathbf{U} : \mathbf{U} \prec X_i\}$, where $\mathbf{U} \prec X_i$ is defined to hold when all nodes in $\mathbf{U}$ precede X_i in $\prec$. The optimal parent set for each node X_i is simply:

$$Pa^*_{\mathcal{G}}(X_i) = \arg \max_{\mathbf{U} \in \mathcal{U}_{i,\prec}} f_D(X_i, \mathbf{U} : D) \tag{2}$$

As the decisions for the different nodes do not constrain each other, this set of selected families provides the optimal network $\mathcal{G}^*_{\prec}$ consistent with $\prec$. Without any restriction on the number of parents for the variables, we note that any acyclic (directed) graph is consistent with some ordering. Hence, the optimal network is simply the network $\mathcal{G}^*_{\prec}$ given by:

$$\prec^* = \arg \max_{\prec} f(\mathcal{G}^*_{\prec} : D) \tag{3}$$

Therefore, the optimal network can be found by searching for the optimal ordering, considering that the score of an ordering is the score of the best network consistent with it. In order to do this, the parent set for each node X_i must be determined. There are two criteria to do this. Thus, the first one consists of looking for the parent set that maximizes f, as it is defined in Sect. 2. This problem is exponential, and there are papers which use an exhaustive search equipped with a parameter k to limit the size of parent sets as in Friedman and Koller (2003) and Teyssier and Koller (2005). On the other hand, the best network can be defined as the minimal *I-map* compatible with the given permutation, that is, we can search for that subset of parents which make the variable X_i conditionally independent of the set of predecessors in the ordering $\mathcal{U}_{i,\prec} \setminus Pa^*_{\mathcal{G}}(X_i)$ given the set of parents $Pa^*_{\mathcal{G}}(X_i)$. In this case, it can be considered that the search is being carried out in the space of *minimal I-maps*.

In this study, we consider both approximations at the same time. In order to do this, the search is carried out with a version of the K2 algorithm which is equipped with an additional deletion operation. At each step, the modified algorithm K2, which will be referred to as *K2M*, analyzes all the possible addition and deletion operations, and chooses independently, for each variable X_i, the one which produces the highest positive difference with respect to f.

K2M depends on two parameters: The ordering $\prec$, and data D. Thus, the resulting graph $\mathcal{G} = K2M(\prec, D)$ is composed of the subgraphs $Pa^*_{\mathcal{G}}(X_i) = K2M(X_i, \mathcal{U}_{i,\prec})$.

This algorithm has already been used in other papers (de Campos and Puerta 2001; Blanco et al. 2003), either as part of a local search algorithm, or with an Estimation of Distribution Algorithm (EDA). However, *K2M* has not been proven yet to be able to find the optimal subset of parents for each variable. Next, the definitions and results ne-cessary to do that are described (Chickering 2002). The following concepts will constitute the theoretical basis of our proposal.

Definition 3 A node or variable X is a *collider* in a path π of $\mathcal{G}$ 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 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 4 Two DAGs are equivalent if they lead to the same *Essential Graph or CPDAG* (Completed Partially Directed Acyclic Graph), that is, if they share the same skeleton and the same v-structures. (Pearl 1988).

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

Definition 6 (*Consistent scoring criterion*)
Let D be a dataset containing m iid samples from some distribution p. Let $\mathcal{G}$ and $\mathcal{H}$ be two DAGs. Then, a scoring metric f is *consistent* if in the limit as m grows large, the following two properties hold:

1. If $\mathcal{H}$ contains p and $\mathcal{G}$ does not contain p, then $f(\mathcal{H} : D) > f(\mathcal{G} : D)$
2. if $\mathcal{H}$ and $\mathcal{G}$ contain p, but $\mathcal{G}$ is simpler than $\mathcal{H}$ (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. This is the basis for defining a criterion score as *consistent*.

Definition 7 (*Locally consistent scoring criterion*)
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 data 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 $f(\mathcal{G} : D) > f(\mathcal{G}' : D)$

Chickering (2002) also proves that the BDe, BIC and MDL scoring criteria are score equivalent, consistent and locally consistent when we suppose that D constitutes a sample which is isomorphic to a graph. The following proposition gives us the way to correctly evaluate (asymptotically) a given order using the *K2M* algorithm:

Proposition 1 *Let D be a dataset containing m iid samples from some distribution p. Let $\hat{\mathcal{G}}$ be the DAG obtained by running the K2M algorithm for each variable X_i, $\hat{\mathcal{G}} = K2M(\prec, D)$. If the metric f used to evaluate DAGs is consistent and locally consistent, then $\hat{\mathcal{G}}$ is a* **minimal I-map** *of p in the limit as m grows large.*

Proof Firstly, we will prove that $\hat{\mathcal{G}}$ is an *I-map* of p. Let us suppose the contrary, i.e., $\hat{\mathcal{G}}$ is not an *I-map* of p. Then there is at least one pair of variables X_i and X_j such that $\langle X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i)\rangle_{\hat{\mathcal{G}}}$ and $\neg I_p(X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i))$. Thus, $\hat{\mathcal{G}}$ cannot be a local optimum of f because the addition of arc $X_j \rightarrow X_i$ has a positive difference.

Finally, we will prove the minimal condition. Again let us suppose the contrary, that is, there exists $X_j \in Pa_{\hat{\mathcal{G}}}(X_i)$ such that $I_p(X_i, X_j | Pa_{\hat{\mathcal{G}}}(X_i) \setminus \{X_j\})$. If so, $\hat{\mathcal{G}}$ cannot be a local optimum because there is (at least) one deletion operation with a positive difference. □

The importance of the result lies in the fact that, given the correct order among the variables, it guarantees under faithful conditions (as the GES algorithm does), that *K2M* would obtain the correct Bayesian network. Moreover, if evaluating a given ordering $\prec$ with the *K2M* algorithm, a minimal *I-map* would be obtained.

3 Local algorithms: HCbO and VNSbO

To define a local method, a neighbourhood operation which transforms one solution to another must be defined. In the case of permutations, used to represent the orderings, it is usual to consider swapping between variables. However, there are alternatives which may allow the optimization of the search procedure. Thus, the algorithms presented in this study are based on the use of an insertion operator which enables a considerable reduction, in the case of local-search-based algorithms, in calculations when building the Bayesian network consistent with each ordering.

The proposed insertion operator is defined as follows: for different indexes i and j, the insert operator modifies the ordering $\prec$, generating a new ordering $\prec'$ by moving the variable in position i to position j in the new ordering. If $j > i$, the variables in positions $(i, j]$ are moved one position **backward** in the ordering. Else, if $j < i$, the variables in positions $[j, i)$ are moved one position **forward** in the ordering. Examples of these definitions can be seen in Figs. 1 and 2.

3.1 The ordering-based Hill-Climbing: HCbO

Once a local operator, such as the previous one, is defined, a greedy Hill-Climbing algorithm based on Orderings (HCbO) can easily be implemented. On each iteration, the algorithm moves from the ordering $\prec_k$ to the ordering $\prec_{k+1}$ by applying the neighbourhood operation which produces the network which the highest positive difference with respect to that obtained from $\prec_k$. The algorithm stops when any neighbour improves on the current one.

The evaluation of all the possible insert operations in an iteration of the HCbO algorithm can be very expensive in terms of computation, since each one produces a new order which must be evaluated, i.e. a network must be built and then f must be calculated. However, the use of the insertion operator makes it possible to carry out some improvements which speed up these operations significantly. These improvements are explained in detail below.

3.1.1 *First improvement*

The first improvement, used in previous studies (Friedman and Koller 2003; de Campos and Puerta 2001; Teyssier and Koller 2005), allows the reduction of the number of computations of parent sets needed in any insert operation *Insert* $(\prec, i, j)$, from n to $|i - j| + 1$. Thus, the computation of the parent sets for those variables which are not located between the variables at positions i and j can be omitted, as the set of variables preceding them is the same in $\prec$ and $\prec'$. Figure 3a shows an example of the situation where this improvement can be applied. As can be seen, the relative ordering for variables A, B, G and H for *Insert*$(\prec, 3, 6)$ is the same after and before the insertion.

3.1.2 *Second improvement*

The second improvement speeds up the computation of some insert operations, *Insert*$(\prec, i, j)$, if the insertion of the variable in position i at another position has already been evaluated. Figure 3b shows the insertion of a variable at different positions. Let us suppose that $\prec' = Insert(\prec, 3, 4)$ has already been calculated, and let us focus on the operation $\prec'' = Insert(\prec, 3, 5)$. Taking into account the first improvement, it is only necessary to recompute the parent set of the variables located between D and C. However, as can be seen in Fig. 3b, the position of the variable D after both insertions is the same. So, it is only necessary to recompute two different parent sets in this operation (E and C).

These two improvements allow an efficient implementation of the insert operation. It is possible to compute only

$$\prec \quad = \quad (\mathbf{X}_1^{\prec}, \ldots, \mathbf{X}_{i-1}^{\prec}, \mathbf{X}_i^{\prec}, \mathbf{X}_{i+1}^{\prec}, \ldots, \mathbf{X}_{j-1}^{\prec}, \mathbf{X}_j^{\prec}, \mathbf{X}_{j+1}^{\prec}, \ldots, \mathbf{X}_n^{\prec})$$

$$\prec' = Insert(\prec, i, j) \quad = \quad (\mathbf{X}_1^{\prec}, \ldots, \mathbf{X}_{i-1}^{\prec}, \mathbf{X}_{i+1}^{\prec}, \ldots, \mathbf{X}_{j-1}^{\prec}, \mathbf{X}_j^{\prec}, \mathbf{X}_i^{\prec}, \mathbf{X}_{j+1}^{\prec}, \ldots, \mathbf{X}_n^{\prec})$$

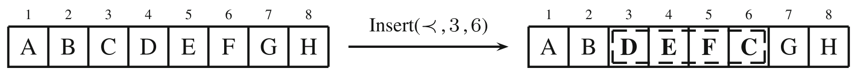


Fig. 1 Definition of the forward insertion and an example

$$\prec \quad = \quad (\mathbf{X}_1^{\prec}, \ldots, \mathbf{X}_{j-1}^{\prec}, \mathbf{X}_j^{\prec}, \mathbf{X}_{j+1}^{\prec}, \;\ldots\; , \mathbf{X}_{i-1}^{\prec}, \mathbf{X}_i^{\prec}, \mathbf{X}_{i+1}^{\prec}, \ldots, \mathbf{X}_n^{\prec})$$

$$\prec' = Insert(\prec, i, j) \quad = \quad (\mathbf{X}_1^{\prec}, \ldots, \mathbf{X}_{j-1}^{\prec}, \mathbf{X}_i^{\prec}, \mathbf{X}_j^{\prec}, \mathbf{X}_{j+1}^{\prec}, \;\ldots\; , \mathbf{X}_{i-1}^{\prec}, \mathbf{X}_{i+1}^{\prec}, \ldots, \mathbf{X}_n^{\prec})$$

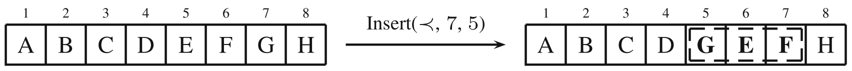


Fig. 2 Definition of the backward insertion and an example

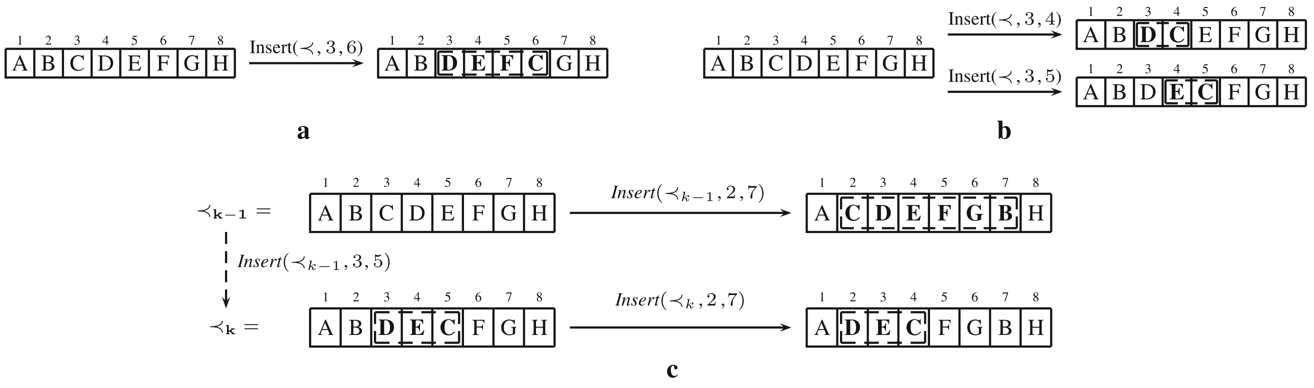


Fig. 3 Example of the improvements (**a** Example of the first improvement, **b** example of the second improvement, **c** example of the third improvement). Only parent sets for the highlighted variables need to be recomputed

two parent sets in each insert operation if they are computed in the correct order, i.e. computing $Insert(\prec, i, j-1)$ before $Insert(\prec, i, j)$ if $j > i+1$ and $Insert(\prec, i, j+1)$ before $Insert(\prec, i, j)$ if $j < i-1$.

3.1.3 Third improvement

The third improvement uses the computations performed in an iteration of the algorithm to skip some computations in the next iteration. In order to clarify the explanation, this improvement will be described individually, without considering the previous improvements. Figure 3c shows an example of the insert operation in two consecutive iterations. The algorithm has moved from $\prec_{k-1}$ to $\prec_k$ using the operation $Insert(\prec_{k-1}, 3, 5)$. In iteration k, for instance, the algorithm needs to compute the operator $Insert(\prec_k, 2, 7)$ (note that $Insert(\prec_{k-1}, 2, 7)$ has already been computed). According to the situation described in the figure, the positions of the variable A and the subset of variables located between F and H are the same in the operations in $\prec_{k-1}$ and $\prec$. Therefore, it is only necessary to compute the new parent set for variables C, D and E, that is, the variables modified by the current insertion but also by the previous accepted insertion. As a direct consequence of this improvement, if the intersection between the previous accepted insertion and the current insertion is null, the computation of the parent sets for all the variables can be omitted.

Analyzing the impact of each one of the proposed improvements on the complexity of the algorithm, it can be seen that the largest reduction is produced by the second improvement, which allows a reduction of order n on average.

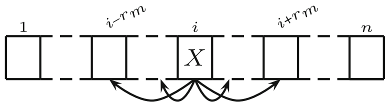


Fig. 4 Behaviour of the maximum radius parameter. Only the insertion of variable X between positions $i - r_m$ and $i + r_m$ is considered

The first improvement makes $\mathcal{O}(n^3)$ calls to the *K2M* algorithm. Since there are n variables which can be inserted in $n-1$ positions and, for each position there are, on average, $n/2$ positions between the indexes used in the insertion operation, the complexity is given by $\mathcal{O}(n \times (n-1) \times n/2)$.

If considering the second improvement, it is only necessary to calculate two new parent sets on each operation, one for each of the affected indexes. Therefore, as there are n variables which can be inserted in $n-1$ positions, and each insertion operation makes only 2 calls to the *K2M* algorithm, the complexity reached with the second improvement is $\mathcal{O}(n \times (n-1) \times 2) = \mathcal{O}(n^2)$.

The third improvement only reduces the variables that are outside the intersection between the current and the previous orderings. Thus, assuming $n/2$ variables in common between the two orderings on average, complexity can be reduced from n to $n/2$, yielding $\mathcal{O}(n \times n/2) = \mathcal{O}(n^2)$.

Even considering the improvements presented, the computation of the insertions could be very expensive. For this reason, the proposed algorithm can use a parameter (named maximum radius) that limits the possible operations. Thus, if a maximum radius r_m is set, for each variable X in position i it only considers the insertion of this variable at positions $j \in [i - r_m, i + r_m]$ (see Fig. 4). This makes it possible to reduce the number of calls to the $K2M$ algorithm, since the neighbours which are not considered are precisely those which need more calls to be evaluated.

The pseudocode for the above method can be seen in Algorithm 1. The function Evaluate($Insert(\prec_k, i, j)$) has implemented the improvements described previously and the function Evaluate($\prec_0$) simply executes $K2M(\prec_0, D)$.

Algorithm 1 HCbO

```
1:  $k = 0$
2:  $\prec_0$= RandomOrdering()
3:  Evaluate($\prec_0$)
4:  loop
5:    for $i = 1$ to $n$ do
6:      for $j = i - 1$ to $\max\{1, i - r_m\}$ do
7:        Evaluate($Insert(\prec_k, i, j)$)
8:      end for
9:      for $j = i + 1$ to $\min\{n, i + r_m\}$ do
10:       Evaluate($Insert(\prec_k, i, j)$)
11:     end for
12:   end for
13:   if $\prec_k^{best}$ is better than $\prec_k$ then
14:     $\prec_{k+1}$=$\prec_k^{best}$
15:     $k = k + 1$
16:   else
17:     break
18:   end if
19: end loop
```

Finally, it is necessary to point out that, as using a maximum radius limits the neighbourhood of an ordering, some orderings cannot be reached in one step. However, it is possible to obtain such orders with two or more intermediate steps.

3.2 VNS based on orderings: VNSbO

Given an ordering, the number of neighbour orderings is directly proportional to the maximum radius. So, for big values of the maximum radius, the number of neighbours that the HCbO needs to evaluate in each iteration can still be high. Therefore, it seems worth starting with a small value for the maximum radius and increasing it during the execution of the algorithm.

The VNS (Mladenović and Hansen 1997) algorithm works with different neighbourhood definitions. In our proposal, called VNSbO, each neighbourhood definition is determined by the value of the current radius (see Fig. 5).

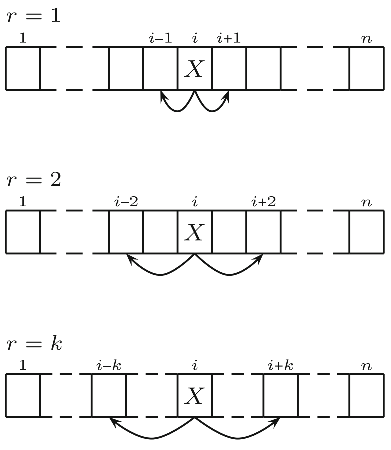


Fig. 5 Radius-based hierarchical neighborhood for VNS algorithm. Variable r defines the neighbourhood. When $r = k$ only the insertion of variable X in positions $i - k$ and $i + k$ is considered

At each iteration of the algorithm, if none of the neighbours of the current ordering improves it, the current radius r is increased by one unit. Otherwise, the algorithm moves from the current ordering to the best neighbour ordering and sets r to 1. The algorithm stops when r is bigger than the maximum radius previously established. If we set r_m equal to $n - 1$ (number of variables minus one) we say that we have a VNSbO without radius restriction.

The pseudocode for VNSbO can be seen in Algorithm 2.

Algorithm 2 VNSbO

```
1:  $k = 0; r = 1$
2:  $\prec_0$= RandomOrdering()
3:  Evaluate($\prec_0$)
4:  while $r \leq r_m$ do
5:    for $i = 1$ to $n$ do
6:      if $i > r$ then
7:        Evaluate($Insert(\prec_k, i, i - r)$)
8:      end if
9:      if $i + r \leq n$ then
10:       Evaluate($Insert(\prec_k, i, i + r)$)
11:     end if
12:   end for
13:   if $\prec_k^{best}$ is better than $\prec_k$ then
14:     $\prec_{k+1}$=$\prec_k^{best}$
15:     $k = k + 1 ; r = 1$
16:   else
17:     $r = r + 1$
18:   end if
19: end while
```

3.3 Iterated versions of HCbO and VNSbO

Considering the behavior of the previous algorithms, there are two aspects which should be mentioned:

- First, given a good initial ordering, the algorithms provide better results and, more importantly, are less time consuming in the execution.
- Second, in order to achieve good initial orderings at early stages in the algorithm, it may not be necessary to be so strict when finding the best set of parents for each variable.

Following the scheme used with the VNS algorithm above, we can consider starting to look for more restricted parents sets in order to find good orderings at these early stages. Then, the complexity of the parent sets can be increased to refine this search. In this study, we propose two iterated algorithms based on HCbO and VNSbO. In these new algorithms, called iHCbO and iVNSbO (Algorithms 3 and 4) respectively, the search will start with a size of one for the parent sets for each variable. Later, when getting stuck in a local optimum, the maximum size of parents sets will be increased by one unit, launching then into another local search. This procedure continues until two consecutive iterations obtain the same Bayesian network. In this case the procedure Evaluate$_s$(*Insert*$(\prec_k, i, i-r)$) is restricted to search parent sets limited to size s.

Algorithm 3 iHCbO

```
1: $k = 0$
2: $s = 1$ // The size of parent sets in the evaluation.
3: $\prec_0 =$ RandomOrdering()
4: loop
5:   Evaluate$_s(\prec_k)$
6:   $changed = FALSE$
7:   loop
8:     for $i = 1$ to $n$ do
9:       for $j = i - 1$ to $\max\{1, i - r_m\}$ do
10:        Evaluate$_s$(Insert$(\prec_k, i, j)$)
11:      end for
12:      for $j = i + 1$ to $\min\{n, i + r_m\}$ do
13:        Evaluate$_s$(Insert$(\prec_k, i, j)$)
14:      end for
15:    end for
16:    if $\prec_k^{best}$ is better than $\prec_k$ then
17:      $\prec_{k+1} = \prec_k^{best}$
18:      $k = k + 1$
19:      $changed = TRUE$
20:    else
21:      $s = s + 1$
22:      break
23:    end if
24:  end loop
25:  if $changed = FALSE$ then
26:    break
27:  end if
28: end loop
```

Algorithm 4 iVNSbO

```
1: $k = 0$; $r = 1$; $s = 1$
2: $\prec_0 =$ RandomOrdering()
3: loop
4:   Evaluate$_s(\prec_k)$
5:   $changed = FALSE$
6:   while $r \le r_m$ do
7:     for $i = 1$ to $n$ do
8:       if $i > r$ then
9:         Evaluate$_s$(Insert$(\prec_k, i, i - r)$)
10:      end if
11:      if $i + r \le n$ then
12:        Evaluate$_s$(Insert$(\prec_k, i, i + r)$)
13:      end if
14:    end for
15:    if $\prec_k^{best}$ is better than $\prec_k$ then
16:      $\prec_{k+1} = \prec_k^{best}$
17:      $k = k + 1$ ; $r = 1$; $changed = TRUE$
18:    else
19:      $r = r + 1$
20:    end if
21:  end while
22:  $s = s + 1$
23:  if $changed = FALSE$ then
24:    break
25:  end if
26: end loop
```

4 Experimental evaluation

In this section, we describe a set of experiments aimed at testing the performance of the algorithms presented in this paper. Below we provide details about the implementation of the algorithms, the datasets used in this comparison, the indicators we chose to argue about the goodness of each algorithm and, to end the section, we give the results and their analysis.

4.1 Experimental setup

In Gámez et al. (2010), the authors compare some of the latest algorithms proposed to carry out the task of learning Bayesian networks. As was remarked in the introduction, the results obtained show that there are some proposals that are more efficient than standard Hill-Climbing in the space of DAGs. However, in terms of accuracy, the compared algorithms are worse than the Hill-Climbing algorithm.

For this reason, in this section, our proposed algorithms are compared with standard Hill-Climbing defined over the space of DAGs (HC), and the Greedy Equivalence Search Algorithm (GES) (Chickering 2002).

The actual implementation includes the improvements presented previously, was coded in Java and interacts with the WEKA library for dataset management. The implementation of HC includes all the common optimizations suggested in the literature. The implementation of GES is based on the one included in Tetrad4.3.9, adds the optimizations suggested in Chickering (2002), and uses the same scoring function as the rest of the algorithms. The GES algorithm is executed without limitation on the maximum number of parents when this is possible.

The score metric used in the algorithms is the Bayesian Dirichlet equivalent in its uniform prior version BDeu (Heckerman et al. 1995). The equivalent sample size (N') used in the experiments is 10, and the network priors are calculated as in Chickering et al. (1995), where $k = 1/(N' + 1)$. As the proposed algorithms are stochastic, we report, for each dataset and algorithm, the execution of 30 independent runs with 30 random initial orderings. For HC and GES the initial DAG is the empty one as usual.

Finally, in order to test the algorithms, we have selected from different sources eight networks to test the algorithms with different sizes and characteristics: ALARM (Beinlich et al. 1989), BARLEY (Kristensen and Rasmussen 2002), CHILD (Cowell et al. 2003), HAILFINDER (Jensen and Jensen 1996), INSURANCE (Binder et al. 1997), MILDEW (Jensen and Jensen 1996), MUNIN version 1 (Andreassen et al. 1989) and PIGS (Jensen 1997), all of them commonly used in the literature. Table 1 shows the main characteristics of these networks. As can be seen, some of them represent domains with few variables, like ALARM and CHILD, whereas others model complex domains with hundreds of variables, such as MUNIN version 1 or PIGS. For each one of the networks we obtained a dataset by sampling 5,000 instances from it. Each dataset was given the same name as its corresponding network.

4.2 Empirical results

To compare the algorithms described in the previous section, we have considered two kinds of factors as performance indicators: the quality of the network obtained by the algorithm, given by the value of the score metric (BDeu) for the resulting model; and the complexity of each algorithm, given by the number of computations of the score metric.

Table 2 shows the BDeu score of the algorithms HC, GES, the unconstrained versions (no radius restriction) of HCbO and VNSbO and also the iterated versions, iHCbO and iVNSbO. The results shown for the database HAILFINDER and the GES algorithm (marked with asterisk) were obtained by limiting the maximum number of parents to 4 (the maximum in degree of the real network). We did this because the algorithm was not able to finish the execution without this limitation. This problem with the GES algorithm is due to variable X_2 that has 1 parent and 16 children such that none of them have any other parent (Fig. 6). For this reason, when transforming the HAILFINDER database to a CPDAG, variable X_2 has 17 undirected edges adjacent to it. In general, in this kind of structures, where a variable has a large number of children that do not have other parents (i.e. a Naive Bayes structure), GES is exponential in the number of variables involved in the structure.

For a more rigorous analysis of the BDeu results in Table 2, we performed a Friedman rank test (Friedman, 1940) as suggested in the literature (Demšar 2006; García and Herrera 2008) to compare the relative performance of multiple algorithms across multiple data sets. Table 3 shows the average rankings of the algorithms calculated in order to perform the Friedman test. With a 95% confidence level, we can reject the hypothesis that all algorithms are equivalent for all eight scenarios. We also performed a post-hoc analysis using Holm's procedure (Holm 1979). The numerical results are shown in Table 4, where Holm's procedure rejects those hypotheses that have a p value ≤ 0.00417. So, according to the test, HC is the worst algorithm, and it presents statistical differences with respect to the rest of the algorithms. The rest of the algorithms do not present statistical differences between them.

A summary of the results for the number of computations of the metric are shown in Table 5. We performed the

Table 1 Bayesian networks used in our experimental evaluation

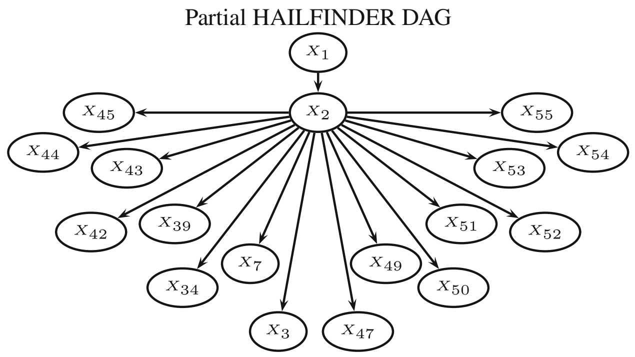
	Variables	Edges	Average states	Min–max states	Max. in–out degree
A-LARM	37	46	2.84	2–4	4–5
BARLEY	48	84	8.77	2–67	4–5
CHILD	20	25	3.00	2–6	2–7
HAILFINDER	56	66	3.98	2–11	4–16
INSURANCE	27	52	3.30	2–5	3–7
MILDEW	35	46	17.60	3–100	3–3
MUNIN 1	189	282	5.26	1–21	3–15
PIGS	441	592	3.00	3–3	2–39

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

Table 2 BDeu score of the networks obtained by the proposed algorithms

	HC	GES	HCbO	VNSbO	iHCbO	iVNSbO
ALARM1	–49,313	–49,163	–48,699	**–48,675**	–48,717	–48,764
BARLEY	–290,677	**–286,309**	–287,162	–287,703	–287,884	–288,503
CHILD	–61,896	**–61,761**	–61,809	–61,868	–61,770	–61,862
HAILFINDER	–253,169	**–253,010***	–253,107	–253,123	–253,159	–253,134
INSURANCE	–68,542	–68,216	**–68,130**	–68,132	–68,159	–68,174
MILDEW	–259,433	**–259,262**	–259,424	–259,439	–259,415	–259,449
MUNIN1	–236,081	–231,365	–223,399	**–223,174**	–224,517	–223,874
PIGS	–1,684,333	**–1,681,903**	–1,682,391	–1,682,379	–1,682,481	–1,682,492

Boldface indicates the best result for each dataset



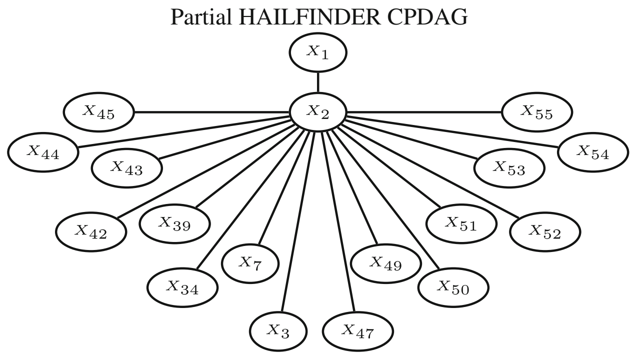


Fig. 6 Partial HAILFINDER DAG and CPDAG. Both graphs were obtained by removing from the complete network variables not in the Markov blanket of variable X_2

Table 3 Average rankings of the algorithms for the BDeu score

Algorithm	Ranking
HC	5.75
GES	2.5
HcbO	2.25
VNSbO	2.75
iHCbO	3.375
iVNSbO	4.375

same tests that we used with the BDeu results in order to compare the complexity of the algorithms. We excluded from this test the algorithm HC because it is statistically worse than the other ones. Including HC in the comparison would add noise as it would be selected as the control algorithm in the test. Table 6 shows the average rankings of the algorithms. The result of the Friedman test is that we can reject the hypothesis that all algorithms are equivalent. Finally, the result of the Holm's procedure is shown in Table 7, where those hypotheses that have a p value ≤ 0.00714 are rejected. The main result obtained is that there is no statistical difference, in terms of the number of computations of the metric, between GES and the algorithms VNSbO, iHCbO and iVNSbO.

Table 4 Statistical analysis of results of Table 2

i	Algorithms	$z = (R_0 - R_i)/\mathrm{SE}$	p	Holm
15	**HC versus HcbO**	**3.74166**	**0.00018**	**0.00333**
14	**HC versus GES**	**3.47440**	**0.00051**	**0.00357**
13	**HC versus VNSbO**	**3.20713**	**0.00134**	**0.00385**
12	**HC versus iHCbO**	**2.53898**	**0.01112**	**0.00417**
11	HcbO versus iVNSbO	2.27172	0.02310	0.00455
10	GES versus iVNSbO	2.00446	0.04502	0.005
9	VNSbO versus iVNSbO	1.73720	0.08235	0.00556
8	HC versus iVNSbO	1.46994	0.14158	0.00625
7	HcbO versus iHCbO	1.20268	0.22910	0.00714
6	iHCbO versus iVNSbO	1.06904	0.28505	0.00833
5	GES versus iHCbO	0.93541	0.34957	0.01
4	VNSbO versus iHCbO	0.66815	0.50404	0.0125
3	HcbO versus VNSbO	0.53452	0.59298	0.01667
2	GES versus HcbO	0.26726	0.78927	0.025
1	GES versus VNSbO	0.26726	0.78927	0.05

Holm's test for $\alpha = 0.05$. Boldface indicates rejected hypothesis

In order to compare our algorithms not only at the end of their execution, but also during the execution, we have also studied the BDeu score and the Structural Hamming Distance (SHD) at each iteration. When the first network is learned, and after each accepted insertion, we measure the BDeu score of the resulting network and the SHD with respect to the original network. The SHD is defined in

Table 5 Number of computations of the score metric

	HC	GES	HCbO	VNSbO	iHCbO	iVNSbO
ALARM1	3,201	5,878	14,268	11,877	8,341	8,336
BARLEY	4,744	9,084	11,080	9,957	8,959	8,954
CHILD	901	3,547	2,352	1,894	1,525	1,503
HAILFINDER	6,541	38,022*	18,580	15,898	13,865	13,427
INSURANCE	1,894	3,195	6,585	5,295	3,916	3,809
MILDEW	2,305	3,507	4,189	3,819	3,418	3,470
MUNIN1	75,766	156,193	307,763	240,963	184,624	175,774
PIGS	527,089	552,732	2,356,271	1,911,619	1,198,004	1,157,968

Table 6 Average rankings of the algorithms for the number of calls

Algorithm	Ranking
GES	2.0
HcbO	3.75
VNSbO	2.75
iHCbO	1.5

Table 7 Statistical analysis of results of Table 5 excluding HC

i	Algorithms	$z = (R_0 - R_i)/\mathrm{SE}$	p	Holm
10	**HcbO versus iVNSbO**	**3.9528**	**0.00008**	**0.0050**
9	**HcbO versus iHCbO**	**3.00416**	**0.00266**	**0.00556**
8	**GES versus HcbO**	**2.84605**	**0.00443**	**0.00625**
7	**VNSbO versus iVNSbO**	**2.68793**	**0.00719**	**0.00714**
6	VNSbO versus iHCbO	1.73925	0.08199	0.00833
5	GES versus VNSbO	1.58114	0.11385	0.01
4	HcbO versus VNSbO	1.26491	0.20590	0.0125
3	GES versus iVNSbO	1.10680	0.26838	0.01667
2	iHCbO versus iVNSbO	0.94868	0.34278	0.025
1	GES versus iHCbO	0.15811	0.87437	0.05

Holm's test for $\alpha = 0.05$. Boldface indicates rejected hypothesis

Complete Partially Directed Acyclic Graphs (CPDAGs) instead of in Directed Acyclic Graphs (DAGs). So, first we transform the networks into their corresponding CPDAGs, and then the SHD 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.

In Fig. 7 we plot the BDeu score and SHD against the number of score computations at each iteration of four representative networks (ALARM, INSURANCE, MUNIN 1 and PIGS) obtained by applying the procedure described in the previous paragraph. The minimum values on the y-axis in the BDeu graphs are adjusted to fit the values obtained by the HCbO and VNSbO algorithms in order to obtain better detail in the rank of BDeu values that are more important. The iterated algorithms usually obtain low BDeu scores in their first steps due to fact that the maximum number of parents is set to small values. For this reason, fitting the y-axis to represent also the first iterations of these algorithms would reduce the resolution of the graphs at the end of the algorithms. The graphs also show the final result obtained by GES and HC. From these graphs the following conclusions can be drawn:

- Usually, as the BDeu score increases, the SHD decreases.
- HCbO performs a large number of computations in the first iteration. By contrast, VNSbO does not have this problem because of the use of a small radius in the initial steps.
- In spite of the fact that iterated algorithms usually obtain low BDeu scores in their first steps, the networks they have learnt when they reach the point (in number of computations) where VNSbO and HCbO find their initial networks, are better.
- In the ALARM, INSURANCE and MUNIN 1 datasets, the networks learnt by the iterated algorithms when they reach the number of computations where GES finishes, are better than the final network obtained by GES.
- There is a common pattern in the graphs with the iHCbO algorithm. First, the BDeu score increases in a small number of computations of the metric, and then, the BDeu score remains at the same value during a large number of computations. This happens when the algorithm increases the maximum number of parents. A new network is learnt with the same ordering, but with the new maximum of parents. The new network usually increases the quality of the previous network. Then, the iHCbO algorithm computes all the possible insert operations. This last step performs a large number of computations of the metric, but the quality of the network is not increased in the process.
- The iVNSbO algorithm presents a similar pattern to the one presented by the iHCbO. It also presents a big improvement of the BDeu score in a small number of computations when the maximum number of parents is increased, but then it does not present a large number of computations without increasing the BDeu score.

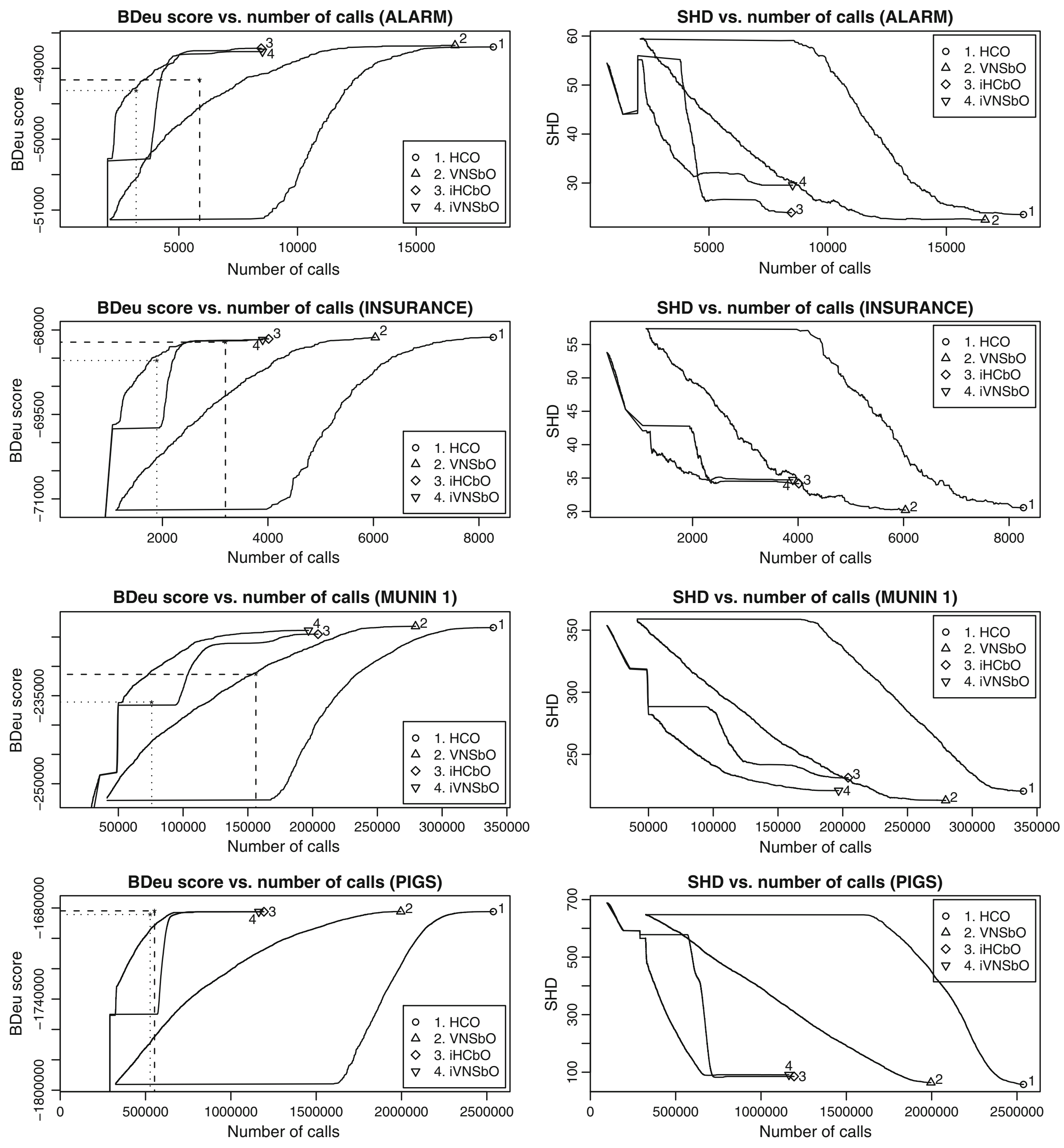


Fig. 7 BDeu score versus number of calls and SHD versus number of calls for the databases ALARM, INSURANCE, MUNIN 1 and PIGS. *Dotted lines* represent the results obtained by HC and *dashed lines* the ones obtained by GES

- In some cases, when the number of parents is increased, the SHD of the newly learnt network increases. This happens because the ordering used to learn the network was a local optimum given the previous maximum number of parents, but this ordering was not good given the new value.

We also performed an analysis of how the maximum radius affects both the accuracy and speed of the algorithms. The maximum radius (r_m) is set proportionally to the number of variables in the dataset using factors 0.2, 0.4 and 0.6. The results are shown in Fig. 8, where we show the graphs of two representative networks, ALARM and

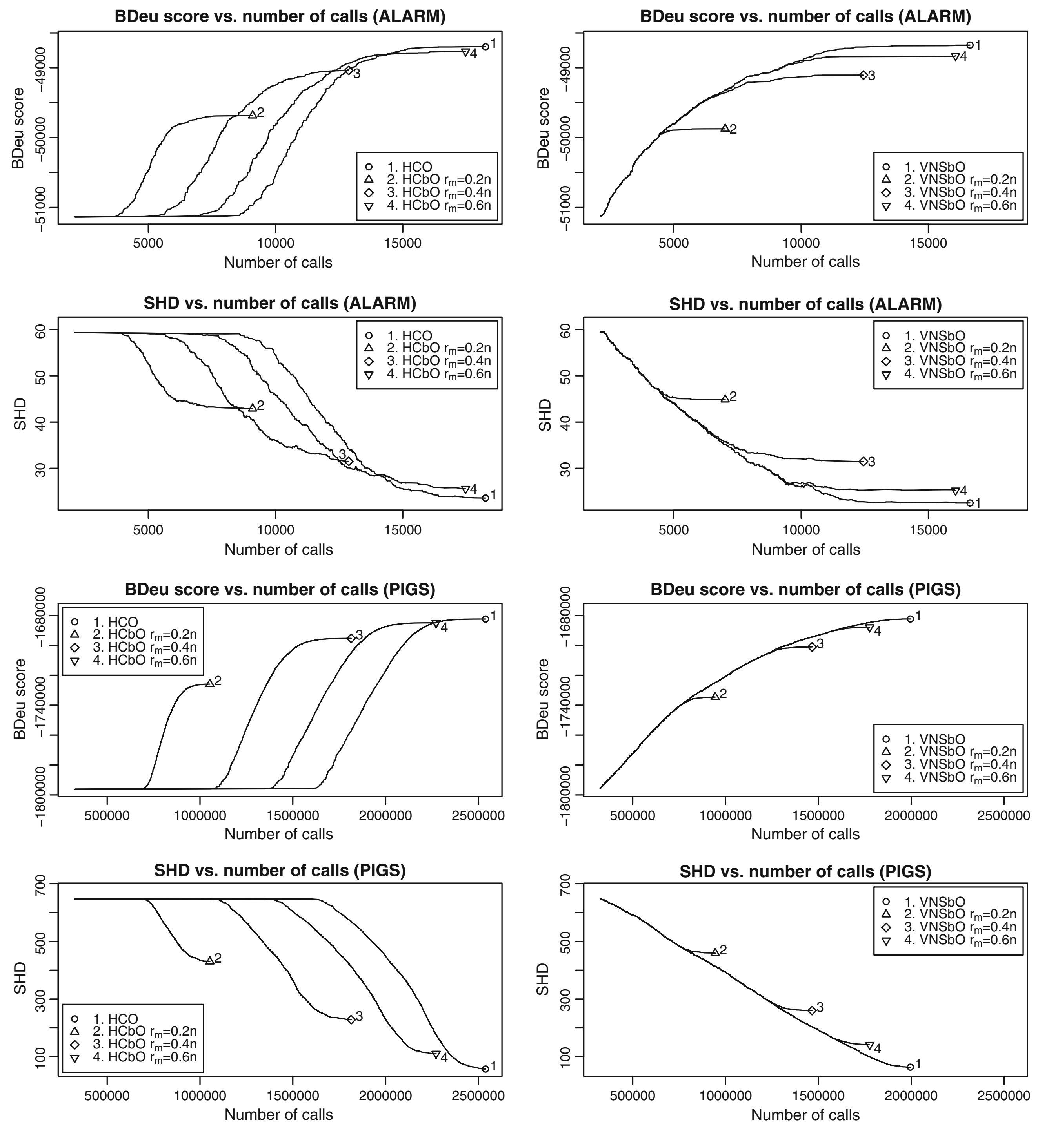


Fig. 8 BDeu score versus number of calls and SHD versus number of calls of the constrained versions of the algorithms for the databases ALARM and PIGS

PIGS. The graphs obtained with the other datasets are similar to the ones included. From this last experiment, we can reach the following conclusions:

- As expected, the final score increases as the maximum radius increases and the number of computations decreases as the maximum radius decreases.
- The number of computations made by the HCbO algorithm in the first iteration can be reduced by constraining the radius, but, in general, the final results obtained with this approach are not good.
- Constraining the radius in VNSbO determines when the algorithm finishes. However, the unconstrained version

of VNSbO outperforms the constrained versions for any given number of score computations. It seems better to run the unconstrained version of VNSbO and stop it when wanted than to use a constrained version.

5 Conclusions and future work

In this paper, we have presented a correct way to evaluate a given ordering when using the space of orderings for structural learning of Bayesian networks, which consists of a modified version of the well-known K2 algorithm equipped with a deletion operation. Moreover, we have introduced a new neighbourhood operator based on insertion as an alternative to the classical swapping operation. This last choice allows improvements in the computations of the neighbourhood, also presented in this paper, which save a lot of parent set computations. We have also presented two local methods based on the previous operator. One is the classical Hill-Climbing search, and the second is a VNS method. This last method can be viewed as the natural extension which automatically adapts the use of the parameter radius during the search. Finally, we have also presented variants of the methods that allow us to reduce the number of computations performed by the algorithms. The variant consists of running the algorithms in an iterated way, increasing the maximum numbers of parents in each run.

From the experiments, we can conclude that HCbO and VNSbO, and their iterated variants iHCbO and iVNSbO, improve on classical HC based on searching the DAGs space in terms of accuracy. Constraining the radius is not the best choice to speed up the algorithms, because the results obtained usually are a little worse. However, the iterated approach speeds up the algorithms and maintains the same performance as the base algorithms. In particular, iHCbO appears to be the best choice considering both complexity and effectivity.

In relation with the GES algorithm, which is considered the reference algorithm for BN learning, the results obtained with the proposed algorithms do not present a significant difference with respect to those obtained with this algorithm in terms of accuracy. In terms of the number of score computations, only HCbO performs significantly more computations than GES. Compared with GES, there are no statistical differences between it and VNSbO, iHCbO or iVNSbO. However, if comparing from a practical point of view, our algorithms are significantly easier to implement. Another problem with the GES algorithm is the need to establish a maximum number of parents in order to assure that the algorithm does not have problems.

In the future, we plan to study how to save more computations in order to speed up the algorithms. This objective can be achieved by searching more restricted neighbourhoods, but assuring that the behaviour of the algorithms is the same. Also, we can use the structure of the BN for a given ordering to characterize the minimal number of operations to be evaluated in the next step of the local algorithm.

References

Andreassen S, Jensen FV, Andersen SK, Falck B, Kjrulff U, Woldbye M, Srensen AR, Rosenfalck A, Jensen F (1989) MUNIN—an expert EMG assistant, computer-aided electromyography and expert systems, chap 21

Beinlich IA, Suermondt HJ, Chavez RM, Cooper GF (1989) The alarm monitoring system: a case study with two probabilistic inference techniques for belief networks. In: Second European Conference on Artificial Intelligence in Medicine, vol 38. Springer, Berlin, pp 247–256

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

Blanco R, Inza I, Larrañaga P (2003) Learning Bayesian networks in the space of structures by estimation of distribution algorithms. Int J Intell Syst 18(2):205–220

Buntine W (1991) Theory refinement on Bayesian networks. In: Proceedings of the seventh conference (1991) on uncertainty in artificial intelligence (UAI'91). Morgan Kaufmann, Los Angeles, pp 52–60

de Campos L, Puerta J (2001) Stochastic local algorithms for learning belief networks: searching in the space of the orderings. In: Symbolic and quantitative approaches to reasoning with uncertainty. Lecture notes in artificial intelligence, vol 2143. Springer, Berlin, pp 228–239

Chickering D (2002) Optimal structure identification with greedy search. J Mach Learn Res 3:507–554

Chickering D, Geiger D, Heckerman D (1995) Learning Bayesian networks: search methods and experimental results. In: Proceedings of the fifth conference on artificial intelligence and statistics, pp 112–128

Chickering DM (1996) Learning Bayesian networks is NP-Complete. In: Fisher D, Lenz HJ (eds) Learning from data: artificial intelligence and statistics V. Springer, Berlin, pp 121–130

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

Cowell RG, Dawid AP, Lauritzen S, Spiegelhalter D (2003) Probabilistic networks and expert systems (information science and statistics). Springer, Berlin

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Friedman M (1940) A comparison of alternative tests of significance for the problem of *m* rankings. Ann Math Stat 11(1):86–92

Friedman N, Koller D (2003) Being Bayesian about network structure: a Bayesian approach to structure discovery in Bayesian networks. Mach Learn 50:95–126

Gámez JA, Mateo JL, Puerta JM (2010) Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. Data Min Knowl Discov (to appear). doi:10.1007/s10618-010-0178-6

García S, Herrera F (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res 9:2677–2694

Heckerman D, Geiger D, Chickering D (1995) Learning Bayesian networks: the combination of knowledge and statistical data. Mach Learn 20(3):197–243

Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6:65–70

Jensen A, Jensen F (1996) Midas—an influece diagram for management of mildew in winter wheat. In: Proceedings of the 12th annual conference on uncertainty in artificial intelligence (UAI-96), pp 349–356

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

Jensen F, Nielsen T (2007) Bayesian networks and decision graphs. Springer, Berlin

Kristensen K, Rasmussen IA (2002) The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides. Comput Electron Agric 33:197–217

Larrañaga P, Poza M, Yurramendi Y, Murga R, Kuijpers C (1996) Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters. IEEE Trans Pattern Anal Mach Intell 18(9):912–926

Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100. http://citeseer.ist.psu.edu/mladenovic97variable.html

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

Teyssier M, Koller D (2005) Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In: UAI '05. Proceedings of the 21st conference in uncertainty in artificial intelligence. AUAI Press, Edinburgh, pp 548–549

Tsamardinos I, Brown LE, Aliferis CF (2006) The max–min hill-climbing Bayesian network structure learning algorithm. Mach Learn 65(1):31–78

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Francisco