On the use of local search heuristics to improve GES-based Bayesian network learning

Juan I. Alonso, Luis delaOssa, José A. Gámez, José M. Puerta
Department of Computing Systems.
Intelligent Systems and Data Mining Group - $i^3\mathcal{A}$.
University of Castilla-La Mancha, 02071 Albacete, Spain.

Abstract

Bayesian networks learning is computationally expensive even in the case of sacrificing the optimality of the result. Many methods aim at obtaining quality solutions in affordable times. Most of them are based on local search algorithms, as they allow evaluating candidate networks in a very efficient way, and can be further improved by using local search-based metaheuristics to avoid getting stuck in local optima. This approach has been successfully applied in searching for network structures in the space of directed acyclic graphs.

Other algorithms search for the networks in the space of equivalence classes. The most important of these is GES (Greedy Equivalence Search). It guarantees obtaining the optimal network under certain conditions. However, it can also get stuck in local optima when learning from datasets with limited size. This article proposes the use of local search-based metaheuristics as a way to improve the behaviour of GES in such circumstances. These methods also guarantee asymptotical optimality, and the experiments show that they improve upon the score of the networks obtained with GES.

This is the accepted version of:

Juan I. Alonso, Luis de la Ossa, Jose A. Gámez, Jose M. Puerta. On the use of local search heuristics to improve GES-based Bayesian network learning. Applied Soft Computing 64:366-376 https://doi.org/10.1016/j.asoc.2017.12.011 (2018)

Please, visit the provided url to obtain the published version.

1 Introduction

Bayesian networks (BNs) [28, 31, 41] are a formalism for knowledge representation that is frequently used in data mining because of the way they manage uncertainty. Concretely, BNs are graphical models that efficiently represent and manipulate n-dimensional probability distributions [41].

The information encoded in a BN is divided into two parts: a *structure*, which is a directed acyclic graph (DAG) representing the qualitative part or relations among variables; and a set of *parameters*, which are locally specified probability distributions representing the quantitative part.

BNs can be used to tackle both descriptive and predictive tasks. Descriptive tasks (identifying dependence relations, clusters, etc.) are carried out by performing *relevance analysis* on the graph [33]. On the other hand, predictive tasks (classification and belief updating/revision) use the (in)dependences codified in the DAG to carry out efficient probabilistic inference [36, 37].

Although BNs can be directly elicited by experts, automatic learning from data is also very usual. There is a vast literature devoted to BN structural learning. However, from an algorithmic point of view, the methods can be divided into two main approaches:

- *Score+Search methods.* They are based on the use of both a scoring function, f, to evaluate candidate structures (DAGs) with respect to the data; and a search method to find the network with the best score.

  Regarding the scoring metrics [39, Chap. 8], they can be either Bayesian (K2 [13], BD [29], etc.), or based on information theory (MDL [6, 32], MIT [18], etc.). As for the search, and due to the NP-hardness of the BN structure learning problem [10], there is a great variety of proposals. Most of them are based on heuristic ([38, 39, 24]), or metaheuristic ([8, 44, 37, 11]) techniques. Some exact algorithms, that return the optimal solution, have been proposed recently [17, 46, 40]. However, they are limited by the size of the problem, as they can only handle a limited number of variables (≤ 35), and require important computational times (the algorithm can take days to compute the result).

- *Constraint-based methods.* These methods are based on finding a network which contains as many of the independences existing in the data as possible [43][39, Chap. 10][45]. They use statistical hypothesis testing to determine the validity of conditional independence sentences.

This paper focuses on the score+search metaheuristic methods.

The way in which the candidate networks are codified affects the design of the methods based on Score+Search, as it determines both the search space and the set of operators that can be used. In this respect, there are two groups of search algorithms. As BNs are represented by DAGs, the most straightforward approach consists of using a DAG to represent each potential solution. This way, the search is carried out in the space of Directed Acyclic Graphs (DAGs) or B-space. On the other hand, it is possible to define *equivalence classes* in the space of DAGs: two DAGs are equivalent if they encode the same probabilistic conditional independences among the variables. Therefore, the search for potential solutions can also be carried out in the space of equivalence classes, or E-space. In general, an equivalence class is represented by means of a hybrid graph, or pattern, with directed and undirected arcs. Although this representation is unique, it might be difficult to manage. However, a unique equivalence class represents several graphs and, more importantly, it is possible to introduce a bias in order to traverse the underlying B-space more efficiently (from the search point of view).

Structural learning of Bayesian networks is computationally expensive. Not only because of the number of candidate solutions that must be evaluated, but also for the cost of each evaluation. For this reason, it is very common to use local search methods, as they allow the evaluation of local changes in the structures, and offer a good trade-off between search efficiency and quality of the networks found.

The simplest local search algorithm defined over the B-space is greedy Hill Climbing (HC) [11]. As starting point for the search, this method takes an arbitrary DAG (usually the empty one). Then, it iteratively inserts or deletes an arc in the current graph (or performs an inversion in some cases) until it is not possible to improve the score. The HC algorithm has been improved by using local metaheuristic schemes to escape from local optima. Some of these methods [16] are Tabu Search (TS) [44], Iterated Local Search (ILS) [20], Variable Neighborhood Search (VNS) [3] and the Greedy Randomized Adaptive Search Procedure (GRASP) [19]. In general, these algorithms improve upon the results obtained by HC, but still remain efficient.

The most important algorithm among those defined over the E-space is GES (Greedy Equivalent Search) [9, 2]. In a similar way to HC, GES iteratively selects the best operation – that is, the insertion or deletion of an arc in the equivalence class represented by the current hybrid graph – until it is not possible to improve the score. GES is the reference algorithm because of its theoretical properties. Thus, if D is a large enough input dataset, sampled from a perfectly Markovian distribution, HC asymptotically converges to a network which represents all the conditional independences present in D,

whereas GES asymptotically converges to the *optimal* network.

Although real data rarely met the aforementioned hypotheses, the results obtained by GES in such cases are equal to, or better than, those obtained by local search-based methods defined over the B-space.

In practice, from a purely optimization perspective, GES can also get stuck at local optima when dealing with real data. Therefore, it can also benefit, in terms of score, from the use of metaheuristics.

The main goal of this paper is to improve the results of GES when learning from data of limited size, where the asymptotic behavior of the algorithm is compromised. There are some related proposals in literature with the same purpose. For example, [15] uses Ant Colony Optimization to learn Bayesian networks in the E-space; whereas [14] uses the so-called Memetic Algorithms. Both works use population-based metaheuristics, which implies the evaluation of a huge number of candidate solutions and, in general, more expensive evaluations – as each structure must be evaluated from scratch. Furthermore, these proposals are not based on GES, but on the representation of equivalence classes. Therefore, do not present asymptotic behavior of GES.

In order to both guarantee the asymptotic behavior of GES, and take advantage of local candidates evaluation, we restrict our approach to local search-based metaheuristics. In particular, to those which have been successfully used in the B-space, such as VNS, ILS and GRASP. In this study, we leave out tabu search, because the theoretical properties of GES cannot be assured with this algorithm.

The rest of the paper is structured as follows. Section 2 briefly revises preliminary concepts and notation relative to Bayesian networks and equivalence classes. Then, Section 3 describes the GES algorithm, which is used as reference in this article, and Section 4 presents the proposed algorithms. The experimental evaluation is described in Section 5. Finally, in Section 6 we summarize our conclusions.

2 Preliminaries

2.1 Bayesian networks

A Bayesian network [41] models the joint probability distribution over a set of variables by using two components:

- A *graphical structure*. Concretely, a DAG $\mathcal{G} = (\boldsymbol{V}, \mathbf{E})$, where $\boldsymbol{V}=$

$\{X_1, X_2, \ldots, X_n\}$ are the nodes (problem domain random variables[1]) and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the set of arcs, which codify the conditional (in)dependence relationships among the variables.

- A set of *numerical parameters* ($\boldsymbol{\Theta}$). For each random variable $X_i \in \boldsymbol{V}$, the BN stores a conditional probability distribution $P(X_i \| \mathbf{Pa}_{\mathcal{G}}(X_i))$, where $\mathbf{Pa}_{\mathcal{G}}(X_i)$ is the parent set[2] of X_i in $\mathcal{G}$. The joint probability distribution can be recovered from the set of conditional distributions by applying the *Markov condition*:

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

  This factorization of the decomposition of the joint probability distribution allows important savings in storage requirements, and makes it possible the design of efficient probabilistic inference algorithms [31].

2.2 Equivalence classes and CPDAGs

Given a DAG $\mathcal{G}$, $\langle \mathbf{X}, \mathbf{Y} \| \mathbf{Z} \rangle_{\mathcal{G}}$ denotes that variables in $\mathbf{X}$ are conditionally independent (through d-separation[3]) of variables in $\mathbf{Y}$ given the set $\mathbf{Z}$. Probabilistic conditional independence regarding distribution p is denoted by $I_p(\mathbf{X}, \mathbf{Y} \| \mathbf{Z})$.

Definition 1 *A DAG $\mathcal{G}$ is an I-map of a probability distribution p if it satisfies that $\langle \mathbf{X}, \mathbf{Y} \| \mathbf{Z} \rangle_{\mathcal{G}} \Rightarrow I_p(\mathbf{X}, \mathbf{Y} \| \mathbf{Z})$. It is a minimal I-map if no arc can be removed from $\mathcal{G}$ without losing the I-map condition.*

Definition 2 *$\mathcal{G}$ is a D-map of p if $\langle \mathbf{X}, \mathbf{Y} \| \mathbf{Z} \rangle_{\mathcal{G}} \Leftarrow I_p(\mathbf{X}, \mathbf{Y} \| \mathbf{Z})$.*

$\mathcal{G}$ is a *perfect-map* of p iff $\mathcal{G}$ is both an I-map and a D-map of p, that is, $\langle \mathbf{X}, \mathbf{Y} \| \mathbf{Z} \rangle_{\mathcal{G}} \Leftrightarrow I_p(\mathbf{X}, \mathbf{Y} \| \mathbf{Z})$. In that case, it is said that p and $\mathcal{G}$ are *faithful* to each other [43, 39]. Furthermore, a distribution p is *faithful* if there exists

[1]Notation: sets and n-dimensional configurations are boldfaced and mathematical structures (graphs, etc.) are in calligraphic font. Capital letters are reserved for variables and sets, while non-capital letters are used for variable values and configurations (vectors).

[2]When a given graph is considered we use $\mathbf{Pa}(X_i)$ to make the notation clear. Since a graph defines a model, all of the usual definitions from the graph theory can be use: nodes, links or edges, directed edges, parent and/or children, the subset of adjacencies of a node, adj(X_i), directed path π, etc.

[3]See [41, Ch. 3] for details.

a graph, $\mathcal{G}$, to which it is faithful. Given any probability distribution p we can obtain a minimal I-map for it. However, in some cases it is not possible to obtain a perfect map for p [41]. In this paper, as in [12] we assume faithfulness, and the terms d-separation and conditional independence will be used interchangeably for p and $\mathcal{G}$.

The goal of BN learning from data is to recover a DAG which is a minimal I-map of the probability distribution encoded by the data. This *relaxation* is due to the fact that we cannot guarantee that a DAG perfectly represents the underlying probability distribution p from which the dataset was sampled.

Two DAGs accounting for the same set of conditional independence relations belong to the same *equivalence class*, and have the property of sharing the same skeleton[4] and the same set of *v-structures*[5]. Given an equivalence class ε, we can distinguish two kinds of arcs:

- *Compelled*, or edges with the same orientation for all DAGs in ε; and
- *Reversible*, or edges whose direction can be different in the DAGs in ε.

Considering this distinction, equivalence classes can be represented by means of hybrid graphs, which contain both directed and undirected arcs. In particular, it is possible to represent an equivalence class by using *Partially Directed Acyclic Graphs* (PDAGs) [12], which do not allow cycles containing only directed arcs. In PDAGs, two nodes are adjacent if they are connected through a directed or undirected arc; and two nodes are neighbors if they are connected by an undirected arc.

A PDAG does not necessarily represent an equivalence class. In fact, only those that allow a *consistent extension*, namely *Completed PDAGs* (CPDAGs), does. A CPDAG is a PDAG whose compelled arcs are oriented, whereas its reversible arcs are not. Given an equivalence class, its corresponding CPDAG is unique [4].

[4]Underlying undirected graph of a DAG.

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

3 Learning BNs by local search methods in the space of equivalence classes

The problem of learning the structure of a BN can be stated as follows: given a training dataset $D = \{\mathbf{v^1}, \ldots, \mathbf{v^m}\}$ of instances of $\mathbf{V}$, find the 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 which evaluates the merit of any candidate DAG with respect to the data $\mathcal{D}$. In general, metrics used in BN structural learning are *decomposable* when there is no missing values. This property allows the computation of the score of a DAG by adding the individual scores assigned to the family of each variable $(X_i \cup Pa_{\mathcal{G}}(X_i))$. Formally,

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

Because of this property, local search methods for learning Bayesian networks are very efficient, as the evaluation of a local operation (which inserts or deletes an edge) only requires recomputing the score of the family (the node X_i and its parent set $Pa_{\mathcal{G}}(X_i)$) involved in the operation.

The Greedy equivalence search (GES) [12] algorithm carries out the search in the space of equivalence classes (instead of in the DAG space). However, it is still considered a local greedy search algorithm, as it uses local operators and also takes advantage of the decomposable metric when evaluating neighbors (candidates).

The search process in GES starts from the equivalence class corresponding to the empty DAG, which represents a probability distribution in which all the variables are marginally independent. Then, it sequentially runs two greedy algorithms. In the first stage, namely *Forward Equivalence Search (FES)*, the algorithm adds one edge at each iteration, stopping when getting stuck in a local maximum. In the second step, namely, *Backward Equivalence Search* (BES), arcs are removed iteratively (also using a greedy approach) from the graph previously obtained by FES. The outcome of GES is an equivalence class which optimizes the scoring metric, and can be written as:

```
GES: return BES(FES(empty network)).
```

Algorithm 3.1 shows the pseudocode for FES. The insertion of an arc in the current equivalence class, ε, is not immediate as in the case of DAGs, and is carefully managed by the *Insert* operator.

Definition 3 (Insert($X, Y, \mathbf{T}$)) *[12]. For non-adjacent nodes X and Y in the current CPDAG ε, and for any subset $\mathbf{T} \subseteq \mathbf{T_0}$ (where $\mathbf{T_0}$ is the set of neighbors of Y that are not adjacent to X), the operator Insert($X, Y, \mathbf{T}$) modifies ε by: (1) inserting the directed edge $X \rightarrow Y$; and (2), for each $T \in \mathbf{T}$, replacing the edge $T - Y$, which is previously undirected, by the directed one, $T \rightarrow Y$.*

Algorithm 3.1 FES

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

10: function FS(ε)
11:     edge = null; best = 0
12:     for all X ∈ V do
13:         for all Y ∈ V | ((Y ≠ X) ∧ (Y ∉ adj(X)) do
14:             for all T ⊆ T0 | (Test(X → Y, T) == true) do
15:                 Compute Δ = Insert(X, Y, T)
16:                 if Δ > best then
17:                     best = Δ; edge = (X → Y); subset = T
18:                 end if
19:             end for
20:         end for
21:     end for
22:     return (X → Y, T, best ) = (edge, subset, best)
23: end function
```

At each iteration, FES adds the candidate arc which maximizes the scoring metric f. The selection of the arc is carried out by the function $FS(\varepsilon)$ (Algorithm 3.1, line 10). FS tests the validity of all possible insertions (Algorithm 3.1, line 14) such that the resulting PDAG, once the arc is added, supports a consistent extension, i.e. the resulting graph is a CPDAG. Then, from among all the insertions passing the test, the algorithm selects the one which produces the best increment in score (Δ) with respect to the current graph ε. Finally, it returns both the arc $X \rightarrow Y$, and the subset $\mathbf{T}$ of variables that produce the best insertion. This process is repeated while f improves (Algorithm 3.1, line 3).

D.M. Chickering [12, Theorem 15 and Corollary 16] describes both the validity tests and the way to locally compute the score of each insertion. The

Algorithm 3.2 BES

```
1: procedure BES(ε)
2:     ε = PDAGtoCPDAG(ε)
3:     (X → Y, H, best ) = BS(ε)
4:     while (X → Y ≠ null) do
5:         ε = Apply Delete(X, Y, H) to ε
6:         ε = PDAGtoCPDAG(ε)
7:         (X → Y, H, best ) = BS(ε)
8:     end while
9:     return ε
10: end procedure

11: function BS(ε)
12:     edge = null; best = 0
13:     for all X ∈ V do
14:         for all Y ∈ V | ((Y ≠ X) ∧ (Y ∈ adj(X)) do
15:             for all H ⊆ H_0 | (Test(X ↛ Y, H) == true) do
16:                 Compute Δ = Delete(X, Y, H)
17:                 if Δ > best then
18:                     best = Δ; edge = (X → Y), subset = H
19:                 end if
20:             end for
21:         end for
22:     end for
23:     return (X → Y, H, best ) = (edge, subset, best)
24: end function
```

PDAG obtained after the insertion might not be a CPDAG. Therefore the operation $PDAGtoCPDAG(\varepsilon)$, which is linear in the number of arcs in ε, is applied. The consistency of this transformation is assured by the validity tests previously performed by the algorithm.

The local optimum/maximum reached by FES, ε, is the starting point for BES, which greedily removes one arc at each step. As happens with insertion in FES, the removal of an arc in BES requires an specific operator:

Definition 4 (Delete($X, Y, \mathbf{H}$)) *[12] For adjacent nodes X and Y in the current CPDAG ε, connected as $X - Y$ or $X \to Y$, and for any subset $\mathbf{H} \subseteq \mathbf{H_0}$ (where $\mathbf{H_0}$ is the set of neighbors of Y that are adjacent to X), the operator Delete($X, Y, \mathbf{H}$) modifies ε by: (1) deleting the edge between X and Y; (2) replacing the edge $H - Y$, which is previously undirected, by a directed one, $Y \to H$, for each $H \in \mathbf{H}$; and (3), replacing any previously undirected edge $H - X$ by a directed one $X \to H$.*

Algorithm 3.2 shows the pseudocode of BES, which makes use of the *Delete* operator to remove arcs. The selection of the best arc to be removed is

Algorithm 3.3 iGES

```
 1: procedure iGES(ε)
 2:     (X → Y, T, bestF) = FS(ε)
 3:     (Z → W, H, bestB) = BS(ε)
 4:     while (X → Y ≠ null) or (Z → W ≠ nul) do
 5:         if (bestF > bestB) then ε = Apply Insert(X, Y, T) to ε
 6:         else ε = Apply Delete(Z, W, H) to ε
 7:         end if
 8:         ε = PDAGtoCPDAG(ε)
 9:         (X → Y, T, bestF) = FS(ε)
10:         (Z → W, H, bestB) = BS(ε)
11:     end while
12:     return ε
13: end procedure
```

also guided by the scoring metric f, and carried out by means of the function $BS(\varepsilon)$ (Algorithm 3.2, line 11). Again, all possible *Delete* operations are tested so as to assure that the resulting PDAG allows a consistent extension. Then, from among the allowed operations, the procedure selects the one which produces the best improvement in f. Finally, it returns both the arc to be removed and $\mathbf{H}$.

[12, Theorem 17 and Corollary 18] describes both the validity tests (Algorithm 3.2, line 14) and the way to efficiently compute the increment in score with respect to the current graph (Algorithm 3.2, line 16) for each deletion.

Lemmas 9 and 10 in [12] demonstrate that the equivalence class obtained by FES is, asymptotically, an I-map of p. Moreover, these lemmas also demonstrate that the equivalence class obtained by BES, and therefore obtained by GES, is asymptotically a perfect map of p. It is important to point out that these theoretical properties require faithfulness to hold.

The author of GES proposed an alternative implementation, namely iGES, which consists of a single stage. At each iteration, it selects the best change – either *Insert* or *Delete*– that can be applied to the current equivalence class ε. The pseudocode of iGES is shown in Algorithm 3.3. Function $FBS(\varepsilon)$ evaluates both the *Insert* and *Delete* operations, and selects the best operation, which is carried out only if it improves the score of ε. After applying each operation, it is necessary to use $PDAGtoCPDAG(\varepsilon)$ to ensure that ε is a CPDAG. This way of implementing GES is fully equivalent to the two-stage one described above, and will be used as the basis for the algorithms proposed in this paper.

Both GES and iGES start the search from the empty graph. However, it is possible to use any other graph ε, provided that it is a CPDAG. Once the algorithm starts with a given CPDAG, the validity test and the insert/delete operations guarantee that the intermediate and the final graphs are correct CPDAGs as well. This is necessary in order to maintain the same theoretical properties than GES.

The assumptions required by GES to guarantee optimality are not often met, as in the real world, the underlying model from which data is *sampled* is not usually a perfect Markovian distribution, and the amount of data available is frequently not large enough. As a consequence, GES usually fails to return the optimal BN.

In this study we analyze the use of standard local search metaheuristics instead of greedy hill climbing as the search method in GES, with the goal of improving the quality of the networks obtained from non-perfect and/or limited data.

4 Using local search metaheuristics to improve GES

Local search methods, such as HC, get stuck at local optima. Thus, if s defines the initial state (configuration or candidate solution) of the search, HC reaches a certain local optimum s^*, usually in a deterministic and memoryless process.

The main reason for using metaheuristics based on local search is to avoid local optima, while preserving the possibilities of efficient local search-based evaluation. The most common strategy consists of iterating a local search function several times, starting from different initial states. The selection of the initial state gives rise to different algorithms, such as Iterated Local Search (ILS), Variable Neighborhood Search (VNS) and the Greedy Randomized Adaptive Search Procedure (GRASP) [26]. This section describes adaptations of iGES that use these local-search-based metaheuristics instead of hill climbing.

The objective of the proposed methods is twofold: firstly, we want to improve upon the results obtained by GES when using general data; and

secondly, we want to guarantee theoretical properties of GES in our proposals. To do this, we have to assure two conditions: (1) the intermediate solutions obtained during the search process must be a CPDAG; and (2) the solution (DAG) obtained must be, under the faithfulness assumption, and asymptotically, a perfect map of p.

–

4.1 Iterated Local Greedy Equivalent Search: ILGES

4.1.1 Iterated Local Search

The simplest approach for avoiding getting stuck at local optima, namely *Random Restart Local Search* (RLS), consists of running HC several times from different randomly-selected starting solutions. However, empirical studies carried out on a large and generic set of problems have shown that the distribution of the score of the solutions found is expected to peak around a solution close in score to the global optimum [34], that is, local optima are often close to each other.

Although a local optimum can contain useful information (implicit in the solution), it is totally discarded in RLS, as each iteration is totally independent of the previous one. The *Iterated Local Search (ILS)* algorithm [34] also restarts HC from different solutions. However, it retains information from one iteration to the next. Thus, once HC reaches a local optimum s^*, ILS obtains the next starting point, s', by perturbing s^*. Afterwards, HC starts a new iteration from s', and reaches a new local optimum, $s^{*\prime}$. Then, if $s^{*\prime}$ is better than s^*, it becomes the next element to be perturbed for generating the next starting solution; otherwise, the algorithm again uses the previous local optimum, s^*.

ILS should lead to good biased sampling, as long as the effect of the perturbation is adjusted. If perturbations are weak, the HC process will often fall back to the starting local optimum. This leads to the exploration of few solutions. On the contrary, as the perturbations become more intense, the new starting point becomes closer to a random one, so there will be no bias in the sampling, and the algorithm will tend to behave like RLS.

4.1.2 Iterated Local Greedy Equivalent Search: ILGES

Iterated Local Greedy Equivalent Search (ILGES) (Algorithm 4.1) is an adaptation of iGES (Algorithm 3.3) which incorporates ILS as search strategy. The use of ILS requires the definition of several components:

- Basic local search algorithm. ILGES uses iGES (Algorithm 3.3).

Algorithm 4.1 Iterated Local Greedy Equivalent Search

```
 1: procedure ILGES
 2:     ε ← Empty network (E = ∅)
 3:     ε* ← iGES(ε); ε ← ε*
 4:     repeat
 5:         ε' ← Pertubation(ε, k)
 6:         ε ← iGES(ε')
 7:         if (ε > ε*) then ε* ← ε
 8:         end if
 9:     until (Termination condition met)
10:     return ε
11: end procedure
12: function Perturbation(ε, k)
13:     it = 0
14:     repeat
15:         rd = U[0, 1]
16:         if rd > 0,5 then
17:             (X → Y, S, best ) = RandomFS(ε)
18:         end if
19:         if rd <= 0,5 then
20:             (X → Y, S, best ) =RandomBS(ε)
21:         end if
22:         ε = Apply Operator(X, Y, S) to ε
23:         ε = PDAGtoCPDAG(ε)
24:     until (k < it++)
25: end function
26: function RandomFS(ε)
27:     ... (same code from Function FS (Alg: 3.1) from line 10 to 15)
28:     Δ = Δ + N(0, 2 × Δ)
29:     ... (same code from Function FS (Alg: 3.1) from line 16 to 23)
30: end function
31: function RandomBS(ε)
32:     ... (same code from Function BS (Alg: 3.2) from line 11 to 16)
33:     Δ = Δ + N(0, 2 × Δ)
34:     ... (same code from Function BS (Alg: 3.2) from line 17 to 24)
35: end function
```

- Perturbation procedure (Algorithm 4.1, lines 5, 12-25). This operator is the key to escape from the local optimum reached in the previous iteration. The current solution is slightly modified in order to move to a new promising starting point in another area of the search space, while retaining enough information from the previous local optimum. Below we detail how the procedure is designed for this particular problem.

  The function Perturbation(ε^*, k) receives two parameters: the state to be altered (a local optimum); and the intensity of the perturbation, which in ILGES corresponds to the number of changes to be applied.

A single change is carried out as follows: first, the type of operator to be used - either arc deletion or insertion- is obtained by random uniform sampling (Algorithm 4.1, lines 16 and 19); afterwards, the algorithm selects the particular application of the operator which, given the current state ε^*, produces the best improvement in score. This step is performed with the functions *RandomFS* and *RandomBS*, which are adaptations of FS and BS (Algorithms 3.1 and 3.2) that use a noisy scoring function. In particular, noise was sampled from a Normal distribution $\mathcal{N}(0, 2s)$, s being the score improvement produced by the selected operation. This way, we introduce randomness into the selection process. RandomFS and RandomBS apply the same validity tests than FES and BES. Therefore, the structure obtained is a valid one, that is, a CPDAG.

Once the best change (insertion or deletion) has been chosen, it is applied regardless of whether the obtained state improves the current one (Algorithm 4.1, line 22) or not. Next, the method applies the conversion to PDAG (Algorithm 4.1, line 23). This guarantees that the graph obtained is a CPDAG after the modification. Lastly, the aforementioned procedure iterates for number of times (k) defined by the user.

- Stopping criterion (Algorithm 4.1), line 9). The algorithm iterates for a number of iterations that is set by the user.

As can be observed, the last iteration of ILGES corresponds to the execution of iGES when taking a valid CPDAG as initial solution, therefore, it is clear that the solution returned by ILGES has the same properties as the one obtained by GES.

4.2 Variable Neighborhood Greedy Equivalent Search: VNGES

4.2.1 Variable Neighborhood Search: VNS

Local methods are based on the definition of neighborhood. That neighborhood is used to traverse the search space by means of local operators which produce small changes in the current state. The Variable Neighborhood Search algorithm [27] uses different nested neighborhood definitions. The search strategy is related to the one described in Subsection 4.1.1. Once the algorithm becomes stuck at a local optimum, it uses a broader neighborhood definition that may allow the algorithm to escape from it. Although there

are several versions of VNS, in this study we only consider the simplest one (*Basic VNS* [27, Section 8.5]), due to the extreme difficulty of systematic exploration of high-order neighborhoods in this particular problem.

The two main components of *Basic VNS* are the local search algorithm and the set of nested neighbourhoods, $\mathcal{N}_k, k = 1, \ldots, k_{max}$.

4.2.2 Variable Neighborhood Greedy Equivalent Search: VNGES

The pseudocode of VNGES is given in Algorithm 4.2. The base local search algorithm is iGES (Algorithm 3.3).

In VNS, the key point to avoid getting trapped in local optima is the use of a set of nested neighborhoods. In the case of VNGES, the $k-th$ neighborhood for a given state ε, $\mathcal{N}_k$, consists of k consecutive applications of a local operator: arc inclusion or arc deletion. Thus, when the algorithm reaches a local optimum in $\mathcal{N}_k$, it generates new neighbors in $\mathcal{N}_{k+1}$.

The function *SelectAtRandomNeighbor(ε, k)* (Algorithm 4.2, line 7) selects the random neighbor by considering the neighborhood $\mathcal{N}_k$. The random selection is based on the scheme followed by the PERTURBATION(.) function described in the ILS algorithm. Therefore, the neighbor is also guaranteed to be a correct CPDAG.

As iGES is used as the basic neighborhood search $k = 1$, and the algorithm guaranties that the final search is initialized with a valid CPDAG, therefore, it is clear that the solution returned by VNGES has the same properties as the one obtained by GES.

4.3 Greedy Randomized Equivalent Search Procedure: GRESP

4.3.1 Greedy Randomized Adaptive Search Procedure: GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP) [42] is a multi-start trajectory-based metaheuristic which has been successfully applied in the literature [35, 7]. In general, it consists of two main phases:

1) Randomized greedy construction.

   - The greedy procedure starts with an empty configuration, and at each step adds the best feature to the current partial solution. This step-wise process stops when the solution is completed.
   - Randomness is introduced in this constructive phase by using a *Restricted Candidate List (RCL)*. At each step the best k features are included in the RCL, and one of them is selected at random.

Algorithm 4.2 Variable Neighborhood Greedy Equivalent Search

```
1: procedure VNGES
2:     Define $\mathcal{N}_k$ and $k_{max}$
3:     $k = 1$
4:     $\varepsilon \leftarrow$ Empty network ($\mathbf{E} = \emptyset$)
5:     $\varepsilon \leftarrow$ iGES($\varepsilon$)
6:     repeat
7:         $\varepsilon^* \leftarrow$ SelectAtRandomNeighbor($\varepsilon, k$)
8:         $\varepsilon' \leftarrow$ iGES($\varepsilon^*$)
9:         if ($\varepsilon' > \varepsilon$) then
10:            $k = 1$
11:            $\varepsilon \leftarrow \varepsilon'$
12:        else
13:            $k = k + 1$
14:        end if
15:    until ($k = k_{max}$)
16:    return $\varepsilon$
17: end procedure
18: function SelectAtRandomNeighbor($\varepsilon, k$)
19:    $\varepsilon' \leftarrow$ Perturbation($\varepsilon, k$) // The function defined in algorithm 4.1
20:    return $\varepsilon'$
21: end function
```

2) Improving phase.

   The solution constructed in the previous stage is, in general, improved by means of a local search-based metaheuristic, normally Hill Climbing.

These two phases are repeated a certain number of times. Although some improvements to this basic scheme have been proposed [22], in this study we focus on the standard version of the GRASP algorithm.

4.3.2 Greedy Randomized Equivalent Search Procedure: GRESP

It is possible to establish a parallelism between the two phases of GES and the two phases of GRASP. The first phase of GES, FES, is a constructive process that adds arcs to the current network (solution) until it finds an I-map. Such an I-map can be considered as a complete solution, as it contains all the dependencies of the model. In the second stage, as BES does in GES, iGES refines the solution obtained by GRES.

Below we describe the two phases of GRESP:

- *Constructive phase.* Our proposal is based on adding randomness to the FES algorithm. The method, namely *Forward Randomized Equivalent Search (FRES)*, is shown in Algorithm 4.3. FRES uses an RCL

Algorithm 4.3 Forward Randomized Equivalent Search (FRES)

```
 1: procedure FRES(ε)
 2:     k = U[2, maxK]
 3:     repeat
 4:         (X → Y, T, best) = FRS(ε,k)
 5:         ε = Apply Insert(X, Y, T) to ε
 6:         ε = PDAGtoCPDAG(ε)
 7:     until (X → Y = null)
 8:     return ε
 9: end procedure

10: function FRS(ε,k)
11:     RCL ← []
12:     for all X ∈ V do
13:         for all Y ∈ V | (Y ≠ X) ∧ Y is not adjacent to X do
14:             for all T ⊆ T_0 | Test(X → Y, T) = true do
15:                 Compute Δ = Insert(X, Y, T)
16:                 if Δ > 0 then
17:                     RCL ← RCL.ADD({X, Y, T}, Δ)
18:                 end if
19:             end for
20:         end for
21:     end for
22:     if RCL == [] then return (null,null,null)
23:     else return (X → Y, T, best) = RCL[U[1, k]]
24:     end if
25: end function
```

of a predefined size, k. At each step, it considers the k best arc additions which pass the validity test (Algorithm 4.3 line 13), and its corresponding subsets $\mathbf{T}$ (function RCL.ADD($\{X, Y, \mathbf{T}\}$, Δ)). From this set of selected arcs, one of them is randomly chosen and added to the current CPDAG model. The constructive process finishes when the RCL is empty, that is, there is no arc addition which improves the current CPDAG. Note that this is the same stopping criterion as in FES, and therefore the same theoretical conditions are retained and the CPDAG returned is an I-map.

- *Improving phase.* The CPDAG constructed by FRES is locally improved by using iGES (Algorithm 3.3).

In our proposal, we introduce a second source of randomness by using a variable value for the size of the RCL at each GRESP iteration. Specifically, the value of k for each GRESP iteration is sampled from $U(2, maxK)$, where $maxK$ is a user defined parameter.

In this case, we can notice that FRES is the main mechanism to: a) trying to escape from the local optima due to the randomness introduced by starting the search in different points; and b) the algorithm offers an IMAP of the distribution, as the original FES, since it stops when no addition of arcs improves the current CPDAG. Once a valid CPDAG has been reached and with no possible improved addition of arcs, the execution of the second phase, BES, is the same as GES and thus the same theoretical properties are guaranteed.

5 Experimental evaluation

5.1 Algorithms

In this section we empirically compare the algorithms presented in this paper against the standard GES algorithm. GES was implemented as described in the original paper. However, we limited the maximum size of the sets **T** and **H**. In practice, as suggested by the author, the size of such sets can be limited without affecting the results [12]. Although in the original paper the authors did not find differences when limiting the size of these sets to 1, we found small differences in big datasets when limiting these sets to sizes 1 or 2. For this reason, we limited the size to 3. This improvement is also implemented in the metaheuristic-based methods.

All the algorithms use BDeu (Bayesian Dirichlet equivalent uniform) [29] as scoring metric, with equivalent sample size $N' = 10$, $\kappa = \frac{1}{(N'+1)}$ and priors computed as in [11]:

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

where D is a dataset; n the number of variables; r_i the cardinality of variable X_i; q_i the number of configurations $Pa(X_i)$ can take; $\Gamma(\cdot)$ stands for the *Gamma* function; N_{ijk} is the number of instances in D with X_i taking its $k-th$ value given the $j-th$ configuration of $Pa(X_i)$; and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

In addition, the proposed metaheuristics need to set some parameters. As the main goal of this work is to study whether the use of metaheuristics can outperform GES, we have not not carried out a fine tuning of these parameters; on the contrary, we use standard values.

In ILGES, k is set to $n/2$ in the PERTURBATION function, and the algorithm stops after 100 iterations. As for VNGES, the maximum neighbourhood size, k_{max}, is set to n. Finally, in GRESP, $maxK$ is set to 8 and the algorithm stops after carrying out 50 iterations.

All the algorithms have been implemented in Java, using `Tetrad 4.3.9`[6] as API. The computational simulations have been ran on quad-core Intel Xenon (3.00 Ghz and 32 Gb of RAM) processors.

5.2 Datasets

This study uses two different benchmarks, and this led to two different experiments.

5.2.1 Benchmark 1: real-world Bayesian networks

This benchmark consists of a set of 9 BNs taken from the Bayesian Network Repository included in the bnlearn R package[7]. We selected 7 networks with *medium* size (20-60 nodes), and two with *large* size (60-100 nodes). Table 1 shows detailed information about the characteristics of such networks: number of variables, edges, values per variable (min, max and mean) and connectivity (max number of parents, children and neighbours (PC[8]).

For each network we independently sampled ten datasets, each one containing 10 000 instances.

5.2.2 Benchmark 2: synthetic Bayesian networks

With the aim of developing a more comprehensive study, we also consider the use of synthetic Bayesian networks. As dealing with a large corpus of datasets increases the input sample for the statistical study, the conclusions drawn will be more robust.

To generate the synthetic networks we have used the procedure introduced in [2, Section 7.3], where three parameters need to be specified: number of variables, n; average number of neighbours per variable, p; and the maximum number of values a variable can take, m.

We consider 8 different patterns for the synthetic networks by combining $n = \{50, 100\}$, $p = \{1, 2\}$ and $m = \{2, 5\}$. Each pattern is denoted by $net(n, p, m)$, e.g., net(50,1,2). Then, 30 networks we generated (sampled) for

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

[7] http://www.bnlearn.com/bnrepository/

[8] Notice that the neighbours of a variable is the union of its parents and children, PC.

Table 1: Real-world networks used in the experiments.

	Brief description	# of nodes	# of edges	average states (min-max)	max parents	max children	max PC
alarm	Monitoring of emergency care patients.	37	46	2.8 (2-4)	4	5	6
barley	Model of barley crop yields.	48	84	8.8 (2-67)	4	5	8
child	Symptoms of disease and birth asphyxia in a child.	20	25	3.0 (2-6)	2	7	4
hailfinder	Predicting hail in northern Colorado.	56	66	4.0 (2-11)	4	16	17
hepar2	Diagnosis of liver disorders.	70	123	2.3 (2-4)	6	17	19
insurance	Evaluating insurance applications.	27	52	3.3 (2-5)	3	7	9
mildew	A model to decide the quantity of fungicide needed to prevent the attack of mildew in wheat.	35	46	17.6 (3-100)	3	3	5
water	A model of the biological processes of a water purification plant.	32	66	3.6 (3-4)	5	3	8
win95pts	A model for printer troubleshooting in Microsoft Windows 95.	76	112	2.0 (2-2)	7	10	10

each pattern by using the procedure described in [2], each one having 5000 instances. Thus, the experiments involved 240 synthetic BNs.

Table 2 shows the characteristics of the synthetic networks. Each row represents the average value over the 30 datasets given a pattern. The values between parentheses are the lower and higher value of the parameter in the set of networks corresponding to that pattern.

Table 2: Synthetic Bayesian networks used in the experiments.

Name	# of vars	# of edges	max parents	max childrens	max PC
net(50,1,2)	50	48.6 (37-59)	4.4 (3-6)	4.3 (3-7)	5.7 (4-8)
net(50,1,5)	50	48.6 (39-59)	4.4 (3-6)	4.2 (3-7)	5.7 (4-8)
net(50,2,2)	50	98.3 (84-118)	6.8 (5-10)	6.7 (5-8)	8.6 (6-12)
net(50,2,5)	50	97.6 (84-118)	6.6 (5-9)	6.7 (5-8)	8.5 (6-12)
net(100,1,2)	100	99.6 (81-121)	5.0 (4-8)	4.6 (3-6)	6.1 (5-8)
net(100,1,5)	100	98.6 (81-116)	4.9 (4-7)	4.6 (3-7)	6.0 (5-7)
net(100,2,2)	100	197.5 (175-222)	7.8 (6-10)	7.4 (5-9)	9.0 (7-11)
net(100,2,5)	100	197.5 (175-221)	7.9 (6-10)	7.6 (5-11)	9.1 (7-11)

5.3 Performance indicators

We base the comparison of the proposed algorithms in two different performance measures:

1. The quality of the network: BDeu score.
2. The resources required by each algorithm: execution time and number of times the scoring metric is called (computed or retrieved).

The use of CPU time as measure has the disadvantage of being dependent on the actual implementation and hardware. On the contrary, counting the number of times that the scoring metric is called has the advantage of being independent of the implementation whilst having a good correspondence with the temporal complexity of the algorithms.

Calls to the scoring metric are cached in order to avoid computing the same function several times, so the number of reported calls has a direct translation to the number of different networks evaluated, giving this way a clue of the search space exploration carried out by the algorithm.

5.4 Results for Benchmark 1: real-world networks

Table 3 shows the results relating to BDeu. The reported results are averaged over the ten independent samples generated for each domain. For each domain the best value is in boldface.

From the results, we can observe that in two datasets (Child and Mildew) all the algorithms obtained the same results, while, for the other 7, GES is always outperformed by the proposed metaheuristics. Specifically, ILGES obtained the best average score in 3 of these datasets; VNGES in 2; and GRESP in the remaining 2.

Tables 4 and 5 show the execution times and the number of calls to the metric, respectively. Besides the results, we also report (in parentheses) the ratio with respect to GES. As expected, the proposed algorithms are slower than GES, and require more calls to the metric. Regarding the difference between the metaheuristics, there is not a clear pattern.

5.4.1 Statistical analysis

For a more rigorous analysis of the results we performed a Friedman rank test [23], as suggested in the literature [21, 25, 5], to compare the relative performance of multiple algorithms across multiple datasets. Table 6 shows the average rankings of the algorithms used to perform the Friedman test. In both dimensions, score and computing requirements, the tests report that at

Table 3: Real networks. BDeu score

Database	GES	GRESP	VNGES	ILGES
Alarm	-106 986	-106 710	-106 678	**-106 670**
Barley	-545 412	-545 216	-544 393	**-544 244**
Child	**-123 201**	**-123 201**	**-123 201**	**-123 201**
Hailfinder	-500 102	**-500 065**	-500 073	-500 077
Hepar2	-327 302	-327 284	**-327 271**	-327 274
Insurance	-134 606	**-133 999**	-134 293	-134 212
Mildew	**-487 043**	**-487 043**	**-487 043**	**-487 043**
Water	-129 503	-129 497	**-129 486**	-129 488
Win95pts	-93 031	-92 954	-92 899	**-92 878**

Table 4: Real networks. Execution time

Database	GES	GRESP		VNGES		ILGES	
Alarm	**10**	422	(42.9)	230	(23.4)	366	(37.2)
Barley	**99**	5 944	(59.9)	7 410	(74.7)	9 045	(91.2)
Child	**8**	150	(19.3)	23	(3.0)	119	(15.4)
Hailfinder	**36**	1 712	(47.5)	2 070	(57.5)	2 951	(82.0)
Hepar2	**27**	1 927	(70.9)	4 414	(162.5)	2 902	(106.8)
Insurance	**7**	397	(59.3)	137	(20.4)	326	(48.7)
Mildew	**42**	5 131	(121.6)	1 097	(26.0)	3 781	(89.6)
Water	**9**	102	(11.8)	55	(6.4)	155	(17.9)
Win95pts	**91**	11 341	(124.9)	17 709	(195.1)	9 563	(105.3)

Table 5: Real networks. Calls to the metric

Database	GES	GRESP		VNGES		ILGES	
Alarm	**5 514**	21 809	(4.0)	23 403	(4.2)	27 605	(5.0)
Barley	**7 678**	22 063	(2.9)	41 733	(5.4)	51 823	(6.7)
Child	**3 473**	8 113	(2.3)	4 918	(1.4)	7 858	(2.3)
Hailfinder	**16 790**	38 378	(2.3)	62 695	(3.7)	75 408	(4.5)
Hepar2	**17 072**	56 862	(3.3)	129 962	(7.6)	121 577	(7.1)
Insurance	**3 277**	13 856	(4.2)	11 016	(3.4)	14 499	(4.4)
Mildew	**3 202**	10 032	(3.1)	14 402	(4.5)	20 157	(6.3)
Water	**3 324**	9 912	(3.0)	16 159	(4.9)	27 263	(8.2)
Win95pts	**60 521**	112 946	(1.9)	255 534	(4.2)	222 572	(3.7)

least one algorithm is different to the others (95% confidence level). Therefore, we ran a post-hoc analysis based on Holm's test [30]. The result shows that, in terms of score, there is only statistically significant difference between VNGES and GES, and IGES and GES (both algorithms outperform GES). With respect to time, GES is significantly different (better) to the other three algorithms. Finally, GES is significantly different (better) to ILGES and VNGES regarding the number of calls to the scoring metric . Detailed results of the statistical tests are shown in Appendix A, Section A.1.

Table 6: Real networks. Ranking of the algorithms (for Friedman test)

	Score	Time	Calls
GES	3.67	**1.00**	**1.00**
GRESP	2.44	2.33	3.11
VNGES	2.00	3.00	2.67
ILGES	**1.89**	3.67	3.22

5.5 Results for Benchmark 2: synthetic networks

The experiments described in the previous section do not allow us to draw convincing conclusions due to the small number of datasets. Thus, from that study some differences are observed between GES and the metaheuristics, but no significant difference is observed among the three metaheuristics, neither in quality nor in efficiency. Therefore, we used the method described in Section 5.2.2 to create a larger corpus (Table 2).

The BDeu results obtained for this new corpus are shown in Table 7, where the number in each cell corresponds to the average score over the 30 networks for each parameter configuration. As can be observed, GES is outperformed by the three metaheuristics in all the cases. In particular, VNGES obtains the best results in 5 cases, ILGES in 2 and GRESP in 1.

Table 8 and Table 9 show the execution time and the number of calls to the metric, respectively, as well as the ratio with respect to GES. As in the case of real networks, GES obtains the best results. If we focus on the metaheuristic algorithms, GRESP requires a small number of calls to the scoring metric, while no differences are observed regarding CPU time. The small number of calls carried out by GRESP with respect to ILGES and VNGES is a sign of a smaller exploration of the search space, hence its worse accuracy.

Table 7: Synthetic networks. BDeu score

Database	GES	GRESP	VNGES	ILGES
net(50,1,2)	-107 313	-107 306	**-107 295**	-107 297
net(50,1,5)	-215 409	**-215 385**	-215 386	-215 386
net(50,2,2)	-111 247	-110 907	-110 926	**-110 899**
net(50,2,5)	-227 833	-227 699	-227 677	**-227 671**
net(100,1,2)	-217 654	-217 634	**-217 589**	-217 594
net(100,1,5)	-440 265	-440 171	**-440 168**	-440 172
net(100,2,2)	-219 735	-219 496	**-219 434**	-219 441
net(100,2,5)	-470 076	-469 899	**-469 847**	-469 847

Table 8: Synthetic networks. Execution time

Database	GES	GRESP		VNGES		ILSGES	
net(50,1,2)	**15**	293	(19.7)	85	(5.7)	580	(39.1)
net(50,1,5)	**9**	362	(39.8)	294	(32.3)	499	(54.8)
net(50,2,2)	**14**	1 813	(129.3)	758	(54.1)	839	(59.9)
net(50,2,5)	**12**	1 031	(84.1)	575	(46.9)	787	(64.2)
net(100,1,2)	**49**	4 257	(87.6)	18 004	(370.4)	9 270	(190.7)
net(100,1,5)	**52**	3 890	(75.1)	12 399	(239.4)	8 743	(168.8)
net(100,2,2)	**158**	33 001	(209.1)	28 700	(181.8)	12 786	(81.0)
net(100,2,5)	**100**	9 696	(96.6)	17 909	(178.5)	8 905	(88.8)

Table 9: Synthetic networks. Calls to the metric

Database	GES	GRESP		VNGES		ILGES	
net(50,1,2)	**8 090**	25 155	(3.1)	59 640	(7.4)	73 109	(9.0)
net(50,1,5)	**7 088**	18 530	(2.6)	51 739	(7.3)	65 141	(9.2)
net(50,2,2)	**9 980**	42 989	(4.3)	53 290	(5.3)	59 552	(6.0)
net(50,2,5)	**8 860**	27 539	(3.1)	49 323	(5.6)	55 216	(6.2)
net(100,1,2)	**45 112**	90 073	(2.0)	493 649	(10.9)	391 024	(8.7)
net(100,1,5)	**29 247**	65 406	(2.2)	401 713	(13.7)	355 774	(12.2)
net(100,2,2)	**42 847**	132 390	(3.1)	419 640	(9.8)	314 548	(7.3)
net(100,2,5)	**35 038**	88 400	(2.5)	347 698	(9.9)	297 222	(8.5)

5.5.1 Statistical analysis

We repeat the same statistical analysis procedure carried out for real-world networks. In all the cases (BDeu, cpu time and number of calls) the Friedman rank test (95% confidence level) rejects the null hypothesis. Table 10 shows the obtained ranks.

From the post-hoc analysis (see Appendix A, Section A.2), the following ranking can be established in terms of BDeu: {VNGES,ILGES} $\succ$ GRESP $\succ$ GES. With respect to the CPU time, a significant difference can be observed for GES with respect to the three metaheuristics algorithms, while no difference at all appears among these. Finally, GES is again the algorithm that requires the smallest number of calls, followed by GRESP, and with VNGES and ILGES in the same (last) group, i.e., GES $\succ$ GRESP $\succ$ {VNGES,ILGES}.

Table 10: Synthetic networks. Ranking of the algorithms (for Friedman test)

	Score	Time	Calls
GES	3.51	**1.00**	**1.02**
GRESP	2.55	2.85	2.01
VNS	**1.91**	3.06	3.51
ILS	2.03	3.08	3.46

6 Conclusions

The most important methodology for the structural learning of BNs is based on searching for the graph (DAG) that best fits the input data according to a scoring metric. This gives rise to the so-called Score+Search methods. The search is usually carried out in the space of DAGs, but it is possible to define the search in the space of equivalence classes, where each configuration joins all the DAGs representing the same set of conditional independences.

The problem of structural learning of BNs, stated as a combinational optimization problem, is NP-hard, and the use of metaheuristics is the common choice to tackle the searching problem. In particular, local search-based methods, due to their efficient evaluation of the candidate solutions and the quality of the solutions obtained.

The GES algorithm is the state of the art for the structural leaning of BNs, and uses greedy local search and the space of equivalence classes. This

algorithm is efficient, and shows competitive solutions when compared with greedy local search algorithms in the DAG space.

The main goal of this paper has been to study the extent to which the GES algorithm is the most suitable by using several classical local-based metaheuristics to show how the results obtained can be improved. We have adapted ILS, VNS and GRASP metaheuristics to search in the equivalence classes using GES as the core local search. We have assured that the new functions included in the phases of the adapted algorithms are theoretically valid to maintain the same properties as GES.

Finally, we have conducted a set of experiments that prove the initial hypothesis of this paper. The proposed algorithms outperform GES, in terms of score, in all the cases. This superiority is validated by means of statistical hypothesis testing in order to extract sound conclusions. However, this performance improvement is produced at the expense of an increase in CPU time. This increase in CPU time is proportional to the number of iterations carried out for the local search used as the core search, GES. This result is not surprising, as it is due to the use metaheuristics. Nevertheless, this increase in running time will be worthwhile as they are offline executions, and their total cost is reasonable. Furthermore, the any-time behavior of the algorithms allow them to be early-stopped if needed. Finally, once we have stated the merit regarding accuracy of the proposed approaches, different strategies for parallelization could be studied (see e.g. [1]), in particular, a coarse-grain parallelization of GRESP is immediate.

As future research, we plan to continue our study by adapting improved versions of ILS, VNS and GRASP and/or by using other local-based techniques, with the goal of improving the quality of the obtained networks but maintaining the competitive time consumption of this kind of algorithms.

Acknowledgments

This work has been partially funded by FEDER funds and the Spanish Government (MINECO) through projects TIN2013-46638-C3-3-P and TIN2016-77902-C3-1-P.

References

[1] Enrique Alba. Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience, 2005.

[2] Juan Ignacio Alonso-Barba, Luis delaOssa, José A. Gámez, and Jose M. Puerta. Scaling up the greedy equivalence search algorithm by constraining the search space of equivalence classes. *Int. J. Approx. Reasoning*, 54(4):429–451, 2013.

[3] Juan Ignacio Alonso-Barba, Luis delaOssa, and Jose M. Puerta. Structural learning of bayesian networks using local algorithms based on the space of orderings. *Soft Comput.*, 15(10):1881–1895, 2011.

[4] Steen A Andersson, David Madigan, and Michael D Perlman. A characterization of markov equivalence classes for acyclic digraphs. *The Annals of Statistics*, 25(2):505–541, 1997.

[5] Jacinto Arias and Javier Cózar. ExReport: Fast, reliable and elegant reproducible research, 2015.

[6] Remco R. Bouckaert. Probabilistic network construction using the minimum description length principle. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, volume 747 of *Lecture Notes in Computer Science*, pages 41–48. Springer, 1993.

[7] Vicente Campos, Rafael Martí, Jesús Sánchez-Oro, and Abraham Duarte. GRASP with path relinking for the orienteering problem. *JORS*, 65(12):1800–1813, 2014.

[8] Xue-Wen Chen, Gopalakrishna Anantha, and Xiaotong Lin. Improving Bayesian network structure learning with mutual information-based node ordering in the k2 algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):628–640, 2008.

[9] David M. Chickering. A Transformational characterization of equivalent Bayesian network structures. In *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 87–98. Morgan Kaufmann, 1995.

[10] David M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

[11] David M. Chickering, Dan Geiger, and David Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 112–128, 1995.

[12] David M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002.

[13] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[14] Carlos Cotta and Jorge Muruzábal. Towards a more efficient evolutionary induction of Bayesian networks. In *PPSN VII Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 730–739, 2002.

[15] Rónán Daly and Qiang Shen. Learning Bayesian network equivalence classes with ant colony optimization. *Journal of Artificial Intelligence Research*, 35(1):391–447, June 2009.

[16] Rónán Daly, Qiang Shen, and Stuart Aitken. Learning Bayesian networks: approaches and issues. *The Knowledge Engineering Review*, 26:99–157, 5 2011.

[17] Cassio P. de Campos and Qiang Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

[18] Luis M. de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7:2149–2187, 2006.

[19] Luis M. de Campos, Juan M. Fernández-Luna, and Jose M. Puerta. Local search methods for learning Bayesian networks using a modified neighborhood in the space of DAG. In *Advances in Artificial Intelligence - IBERAMIA 2002, 8th Ibero-American Conference on AI, Seville, Spain, November 12-15, 2002, Proceedings*, pages 182–192, 2002.

[20] Luis M. de Campos, Juan M. Fernández-Luna, and Jose M. Puerta. An iterated local search algorithm for learning Bayesian networks with restarts based on conditional independence tests. *Int. J. Intell. Syst.*, 18(2):221–235, 2003.

[21] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[22] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[23] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[24] José A. Gámez, Juan L. Mateo, and José 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)):106–148, 2011.

[25] Salvador García and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, December 2008.

[26] Fred Glover and Gary A. Kochenberger, editors. *Handbook of metaheuristics*. International series in operations research & management science. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003.

[27] Pierre Hansen and Nenad Mladenović. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Variable Neighborhood Search, pages 211–238. Springer US, 2005.

[28] David Heckerman. Bayesian networks for data mining. *Data Mining and Knowledge Discovery*, 1:79–119, 1997.

[29] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[30] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[31] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs (2ed)*. Springer Verlag, 2007.

[32] Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–294, 1994.

[33] Yan Lin and Marek J. Druzdzel. Computational advantages of relevance reasoning in Bayesian belief networks. In *In Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 342–350. Morgan Kaufmann, 1997.

[34] HelenaR. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363–397. Springer US, 2010.

[35] Rafael Martí, Vicente Campos, Mauricio G. C. Resende, and Abraham Duarte. Multiobjective GRASP with path relinking. *European Journal of Operational Research*, 240(1):54–71, 2015.

[36] Denis D. Mauá, Cassio P. de Campos, Alessio Benavoli, and Alessandro Antonucci. Probabilistic inference in credal networks: New complexity results. *J. Artif. Intell. Res. (JAIR)*, 50:603–637, 2014.

[37] Pedro Larrañaga, Hossein Karshenas, Concha Bielza, and Roberto Santana. A review on evolutionary algorithms in Bayesian network learning and inference tasks. *Information Sciences*, 233:109 – 125, 2013.

[38] Andreas Nägele, Mathäus Dejori, and Martin Stetter. Bayesian substructure learning - approximate learning of very large network structures. In *Proceedings of the 18th European conference on Machine Learning (ECML '07)*, pages 238–249, 2007.

[39] Richard Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[40] Pekka Parviainen and Mikko Koivisto. Finding optimal Bayesian networks using precedence constraints. *Journal of Machine Learning Research*, 14(1):1387–1415, 2013.

[41] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference*. Morgan Kaufmann, 1988.

[42] Mauricio G. C. Resende and Celso C. Ribeiro. *Handbook of Metaheuristics*, chapter Greedy Randomized Adaptive Search Procedures, pages 219–249. Springer US, 2003.

[43] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction and Search*, volume 81 of *Lecture Notes in Statistics*. Springer Verlag, 1993.

[44] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

[45] Raanan Yehezkel and Boaz Lerner. Bayesian network structure learning by recursive autonomy identification. *Journal of Machine Learning Research*, 10:1527–1570, 2009.

[46] C. Yuan and B. Malone. Learning optimal bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–25, 2013.

A Statistical analysis

A.1 Real networks

Table 11: Real networks. BDeu score. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
ILGES vs VNS	4/2/3	1.0
ILGES vs GRESP	5/2/2	1.0
VNGES vs GRESP	5/2/2	1.0
GRESP vs GES	7/2/0	1.78e-01
VNGES vs GES	7/2/0	**3.08e-02**
ILGES vs GES	7/2/0	**2.09e-02**

Table 12: Real networks. Execution time. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
VNGES vs GRESP	5/0/4	1.0
VNGES vs ILGES	7/0/2	1.0
GRESP vs ILGES	4/0/5	1.0
GES vs VNGES	9/0/0	**2.46e-02**
GES vs GRESP	9/0/0	**2.61e-03**
GES vs ILGES	9/0/0	**1.57e-03**

Table 13: Real networks. Calls to the metric. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
GRESP vs VNGES	7/0/2	5.47e-01
VNGES vs ILGES	7/0/2	5.47e-01
GRESP vs ILGES	8/0/1	1.14e-01
GES vs GRESP	9/0/0	1.14e-01
GES vs VNGES	9/0/0	**5.08e-03**
GES vs ILGES	9/0/0	**7.06e-05**

A.2 Synthetic networks

Table 14: Synthetic networks. BDeu score. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
VNGES vs ILSGES	81/101/58	3.31e-01
ILSGES vs GRESP	120/77/43	**1.55e-05**
VNGES vs GRESP	125/76/39	**1.56e-07**
GRESP vs GES	138/101/1	**2.62e-15**
ILSGES vs GES	173/67/0	**1.96e-35**
VNGES vs GES	173/67/0	**6.83e-41**

Table 15: Synthetic networks. Execution time. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
VNGES vs ILGES	116/0/124	8.60e-01
GRESP vs VNGES	131/0/109	1.55e-01
GRESP vs ILGES	144/0/96	1.55e-01
GES vs GRESP	240/0/0	**3.59e-55**
GES vs VNGES	240/0/0	**7.05e-68**
GES vs ILGES	240/0/0	**3.74e-69**

Table 16: Synthetic networks. Calls to the metric. Adjusted p-values of the pairwise-based post-hoc Holm's test

	win/draw/lose	p_value
ILGES vs VNGES	127/0/113	6.97e-01
GES vs GRESP	236/0/4	**5.83e-17**
GRESP vs ILGES	238/0/2	**2.60e-34**
GRESP vs VNGES	235/0/5	**2.60e-36**
GES vs ILGES	240/0/0	**5.69e-95**
GES vs VNGES	240/0/0	**1.94e-98**