# New Local Move Operators for Learning the Structure of Bayesian Networks

**Jimmy Vandel** and **Brigitte Mangin** and **Simon de Givry**
**UBIA, UR 875, INRA, F-31320 Castanet Tolosan, France**
**{jvandel,mangin,degivry}@toulouse.inra.fr**

**Abstract.** We propose new local move operators incorporated into a score-based stochastic greedy search algorithm to escape more effectively from local optima in the search space of directed acyclic graphs. We extend the classical set of arc addition, arc deletion, and arc reversal operators with a new operator replacing or *swapping* one parent to another for a given node, *i.e.* combining two elementary operations (arc addition and arc deletion) in one move. The old and new operators are further extended by doing more operations in a move in order to overcome the acyclicity constraint of Bayesian networks. These extra operations are temporally performed in the space of directed *cyclic* graphs, such that at the end, they recover graph acyclicity and increase the score. Our experimental results on standard Bayesian networks and challenging gene regulatory networks show large BDeu score and recall value improvements compared to state-of-the-art structure learning algorithms when the sample size is small.

## 1 Introduction

Learning the structure of Bayesian networks from fully observed data is known to be an NP-hard problem [6] which has received a lot of attention from researchers during the last two decades [9]. Due to its difficulty, heuristic methods have been widely used to learn Bayesian network structures. Two main approaches have been studied: constraint-based methods and score-based methods [19]. Constraint-based methods aim at satisfying as much independence present in the data as possible using conditional independence tests. Unfortunately, these methods can be sensitive to failures in individual independence tests. Score-based methods use a scoring function $f$ to score a network structure with respect to the data. They score the whole network at once, being therefore less sensitive to individual failures. Different Bayesian and non-Bayesian scoring metrics can be used such as the *Bayesian Information Criterion* (BIC) [27] or the *Bayesian Dirichlet criterion* (BDeu) [3, 17]. Score-based methods explore the space of network structures to find the highest-scoring network. This space being superexponential, local search methods are used such as greedy ascent search (also called hill-climbing), tabu search, simulated-annealing, and other complex metaheuristics like ant colony optimization [8]. In spite of its simplicity, the (repeated randomized or *stochastic*) greedy search method reveals to be a competitive method compared to more complex algorithms [13]. Starting from an initial network structure, it performs a series of local moves until a local optimum is found. Each move selects and applies the best elementary operation(s) on the current network structure. The set of candidate neighboring structures is called the *neighborhood* in the sequel. A classical neighborhood is composed of single arc additions, deletions, and reversals. Using *larger* neighborhoods efficiently allows to find better local optima, and thus better network structures.

[21] proposed an optimal reinsertion of a target node by removing all its edges and reinserting it optimally. However this approach is limited to small problems only. [18] used a restricted form of look ahead called LAGD, combining several operations in a single move. In this paper, we follow a similar approach by focusing our local operations on a target node and combining several operations guided by the global acyclicity constraint of Bayesian networks. By doing so, we are able to exploit large neighborhoods efficiently. Other approaches use a compact representation of a set of network structures. They explore either the search space of variable orderings (an optimal structure compatible with the order being easier to find) [7, 29, 26, 2], or the search space of Markov-equivalent network structures like *Greedy Equivalence Search* (GES) [4, 22, 8].

In Section 2, we give Bayesian network background. Next, we define a specific stochastic greedy search algorithm and introduce the new local move operators in Section 3. We report experimental results in Section 4 and conclude.

## 2 Bayesian network structure learning

A Bayesian network [19] denoted by $B = (G, \mathbf{P}_G)$ is composed of a directed acyclic graph (DAG) $G = (\mathbf{X}, \mathbf{E})$ with nodes representing $p$ random discrete variables $\mathbf{X} = \{X_1, \ldots, X_p\}$, linked by a set of directed edges or *arcs* $\mathbf{E}$[1], and a set of conditional probability distributions $\mathbf{P}_G = \{P_1, \ldots, P_p\}$ defined by the topology of the graph: $P_i = \mathbb{P}(X_i|Pa(X_i))$ where $Pa(X_i) = \{X_j \in \mathbf{X} \,|\, (X_j \rightarrow X_i) \in \mathbf{E}\}$ is the set of parent nodes of $X_i$ in $G$. A Bayesian network $B$ represents a joint probability distribution on $\mathbf{X}$ such that: $\mathbb{P}(\mathbf{X}) = \prod_{i=1}^{p} \mathbb{P}(X_i|Pa(X_i))$.

Conditional probability distributions $\mathbf{P}_G$ are determined by a set of parameters. Given the network structure $G$, and the fully observed data $D$, parameters can be estimated by simple counting, following the maximum likelihood principle.

Learning the structure of a Bayesian network consists in finding a DAG $G$ maximizing $\mathbb{P}(G|D)$. We have $\mathbb{P}(G|D) \propto \mathbb{P}(D|G)\mathbb{P}(G)$. Under specific assumptions, the *marginal loglikelihood* $\log(\mathbb{P}(D|G))$ can be expressed as a decomposable scoring function, such as the BIC and BDeu criteria:

$$f(D, G) = \sum_{i=1}^{p} f_{X_i}(D, G) = \sum_{i=1}^{p} f_{X_i}(D, Pa(X_i)) \tag{1}$$

[1] In the paper, we use $G = \mathbf{E}$ when the set of nodes is implicit.

A set of Bayesian networks are Markov-equivalent if they imply exactly the same set or *map* of independence constraints among variables[2]. Next, we describe a novel greedy search method maximizing $f$ in the space of DAGs.

## 3 Stochastic Greedy Search

We define the Stochastic Greedy Search (SGS) algorithm[3] for structural learning of Bayesian networks in Algorithm 1. It collects the best DAG found by $r$ randomized hill climbing algorithms. Stochasticity comes from two random draws. The first, that is commonly seen in the structure learning literature, is due to `InitGraph` that returns a random DAG used by the inner greedy search loop. The second is more original. It is the random DAG drawn within the best neighbors in the neighborhood of the current DAG $G$ ( `SelectRandom` at line 1). The neighborhood of $G$, returned by `Neighbors`, is composed of the usual elementary operations on DAGs: arc addition (ADD), arc deletion (DELETE), and arc reversal (REVERSE). This classical neighborhood is denoted $\mathcal{N}^{ADR}$ in the sequel. Only feasible operations are considered, which do not create cycles. In the next subsections, we are going to extend this set of operations. We show also in the experiments that the first random draw may be counter-productive and starting from an empty graph is often better than from a random DAG, as also observed in [20].

**Algorithm 1**: Stochastic Greedy Search algorithm.

```
Input  : An iid sample D, a scoring function f, integer r
Output : A directed acyclic graph
G* ← ∅ /* Best DAG initialized with the empty graph */;
s* ← f(D, G*) /* score of the empty graph */;
/* Repeat r randomized greedy searches */;
for i ← 1 to r do
    G ← InitGraph() /* Choose an initial DAG */;
    s ← f(D, G) ;
    repeat
        improvement ← false ;
        s^max ← max_{G' ∈ Neighbors(G)} f(D, G');
        if s^max > s then
            /* Select at random among the best neighbors */;
            G^max ← {G' ∈ Neighbors(G) | f(D, G') = s^max} ;
1           G ← SelectRandom(G^max) ;
            s ← s^max ;
            improvement ← true ;
    until ¬ improvement ;
    /* Keep the best DAG of r greedy searches */;
2   if s > s* then
        G* ← G ;
        s* ← s ;
return G* ;
```

**Proposition 1.** *Let $D$ be a dataset of $n$ records that are identically and independently sampled from some distribution $\mathbb{P}(\cdot)$. Let $f$ be a locally consistent scoring function. The inner loop of SGS returns a minimal independence map of $\mathbb{P}(\cdot)$ as the sample size $n$ grows large.*

Recall that BIC and BDeu scores are locally consistent [4]. The local consistency property ensures that adding any arc that eliminates an independence constraint that does not hold in the generative distribution $\mathbb{P}(\cdot)$ increases the score. Conversely, deleting any arc that results in a new independence constraint that holds in $\mathbb{P}(\cdot)$ also increases the score. Hence, starting from any DAG, by selecting strictly improving moves, the algorithm terminates (because of a finite number of DAGs) and returns an independence map of $\mathbb{P}(\cdot)$ (*I-map*: every independence implied by the resulting DAG is verified in $\mathbb{P}(\cdot)$, in the worst case it can be a complete graph) which is minimal thanks to arc deletion operations and local consistency of $f$.

The main interest of our randomization approach is to simulate a search in the space of score-equivalent networks. Each greedy search moves from a randomly-selected DAG instance of a Markov-equivalence class $\mathcal{E}(G)$ to another randomly-selected DAG of an *adjacent*[4] Markov-equivalence class $\mathcal{E}(G')$ thanks to our `SelectRandom` function. It results in a stronger property:

**Proposition 2.** *Let $D$ be a dataset of $n$ fully observed records that are an* iid *sample of some* faithful *distribution $\mathbb{P}(\cdot)$. Let $f$ be a locally consistent scoring function. SGS returns a perfect map of $\mathbb{P}(\cdot)$ as both the sample size $n$ and the number of restarts $r$ grow large.*

Recall that a faithful distribution admits a unique perfect map corresponding to the optimal structure. Compared to the GES algorithm [4], which offers the same optimality guarantee within a two-phase greedy search, SGS chooses the orientation of some *compelled arcs*[5] of the *true* DAG at random, whereas GES waits while no *v-structures* impose orientation constraints. See an example in Figure 1.

Notice that neither GES nor SGS find an optimal structure in polynomial time in the worst case, even when using a constant time consistent scoring function and a faithful distribution[6]. In general, without the faithfulness assumption, learning the optimal structure is NP-hard even when the sample size is large and when each node has at most $k$ parents, for all $k \geq 3$ [5].

In the experiments, we show that a small number of restarts $r$ allows to find DAGs with better scores than GES, especially when the sample size $n$ is limited. In this case, GES found a local optimum and SGS is able to find other better local optima thanks to randomization, being not totally greedy in the space of score-equivalent networks. This was also observed in [22].

When the sample size is small the learning problem becomes more difficult: the empirical distribution may be far from a perfect map resulting in many local optima and the scoring function is no more consistent, *i.e.* the likelihood does not dominate the penalty term which is a non additive function of the parent variable domain sizes [4]. In this complex situation, we propose a new operator to escape from some local optima.

### 3.1 SWAP operator

Consider the 3-variable example in Figure 2 with observed data $D$, scoring function $f$, and initial DAG $G_0 = \{X2 \rightarrow X3\}$. Let assume $f(D, \{X1 \rightarrow X3\}) > f(D, \{X2 \rightarrow X3\}) > f(D, \{X1 \rightarrow X3, X2 \rightarrow X3\}) > f(D, \{X3 \rightarrow X1, X2 \rightarrow X3\}) > f(D, \{X2 \rightarrow X1, X2 \rightarrow X3\}) > f(D, \emptyset)$. Then $G_0$ is a local minimum for the classical neighborhood $\mathcal{N}^{ADR}$. Our new operator, denoted SWAP($X|Y \rightarrow Z$), consists in changing one parent $X$ to another parent $Y$ for one target node $Z$. This is equivalent to a simultaneous pair of ADD and

[2] BIC/BDeu give the same score for Markov-equivalent DAGs.

[3] Don't mistake for the SGS algorithm in [28] which is a constraint-based algorithm, but ours is score-based.

[4] Two equivalence classes $\mathcal{E}(G)$, $\mathcal{E}(G')$ are adjacent iff $G$ is an I-map of $G'$ or vice-versa and the number of edges in the graphs $G$ and $G'$ differs by one.

[5] An arc $X \rightarrow Y$ in $G$ is *compelled* if that arc exists in every DAG of $\mathcal{E}(G)$, otherwise it is said *reversible*.

[6] The number of possible sets $S$ in the GES operator ADD($E$,$S$) is exponential in the maximum degree $d$ of the current graph in the worst case and $r$ is unbounded for SGS.

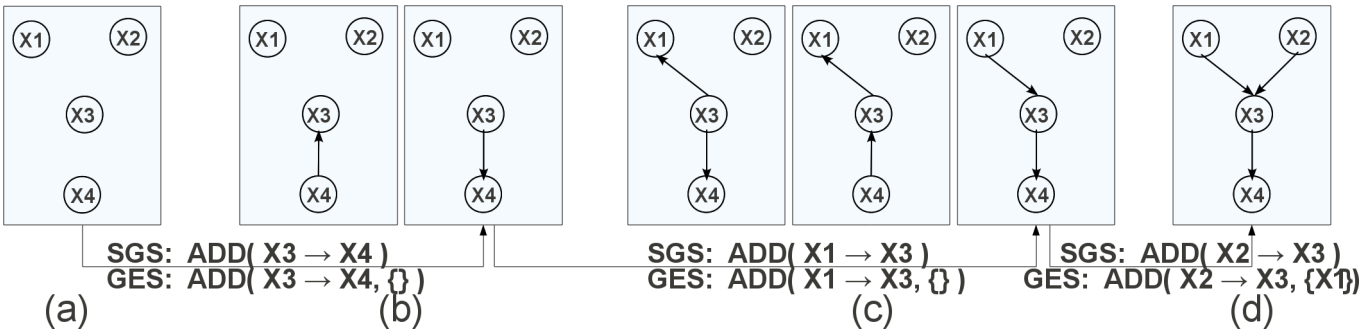


**Figure 1.** Four adjacent Markov-equivalence classes found by GES during its first phase of edge and v-structure insertions. (a) GES and SGS start from the empty graph . (d) The true DAG is found after three moves. The orientation of $X3 \rightarrow X4$ and $X1 \rightarrow X3$ edges are chosen at random by SGS, whereas GES waits until its third move to decide on edge orientations based on DAG score comparisons (enforcing the v-structure $X1 \rightarrow X3 \leftarrow X2$ as stated by the extra ADD parameter $\{X1\}$, and forbidding $X1 \rightarrow X3 \leftarrow X4$ in its second move).

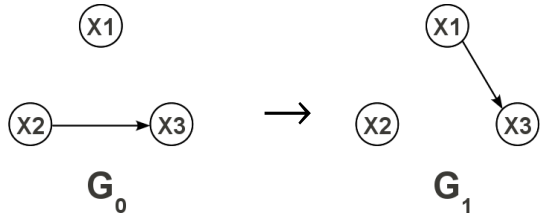


**Figure 2.** The operator SWAP($X2|X1 \rightarrow X3$) applied to a 3-variable problem.

DELETE operators restricted to the same target node. In our example, applying SWAP($X2|X1 \rightarrow X3$) corresponds to DELETE($X2 \rightarrow X3$),ADD($X1 \rightarrow X3$), resulting in the better DAG $G_1 = \{X1 \rightarrow X3\}$ as shown in Figure 2. The extended neighborhood using the four operators is denoted $\mathcal{N}^{ADRS}$ and SGS using $\mathcal{N}^{ADRS}$ (respectively $\mathcal{N}^{ADR}$) is denoted SGS$^2$ (Stochastic Greedy Search with Swap) (resp. SGS$^1$) in the sequel.

A typical suboptimality problem that we observed in our experiments happens when two nodes have the same parents. Figure 3 shows such an example with four variables. Because child nodes $X3$ and $X4$ are highly correlated due to their common parents $X1$ and $X2$, the first arc usually added is either $X3 \rightarrow X4$ or $X4 \rightarrow X3$ especially if the conditional probability distributions of $X3$ and $X4$ given $X1$ and $X2$ are close. Then adding a first parent to $X3$ and furthermore a second parent to $X4$ and $X3$ give the DAG $G_4$. Here deleting $X3 \rightarrow X4$ is going to decrease the score and the same negative effect results when adding $X1 \rightarrow X4$ because it leads to a three-parent node. For this node, the increase in likelihood may not overcome the penalization term. But doing both operations simultaneously thanks to our SWAP operator results in a better local optimum DAG.

Let $p$ be the number of variables in the current DAG and $k$ be the maximum number of parents per node. Assuming a sparse graph, $p \gg k$, the number of SWAP operations is bounded by $O(kp^2)$ in the extended neighborhood $\mathcal{N}^{ADRS}$, whereas it is bounded by $O(p^2)$ for ADD and $O(kp)$ for DELETE and REVERSE. The complexity of $\mathcal{N}^{ADRS}$ is therefore in $O(kp^2)$, whereas it is in $O(p^2)$ for the classical neighborhood $\mathcal{N}^{ADR}$. Notice that other approaches using larger neighborhoods such as *h-look ahead in l good directions* (LAGD) has a worst-case complexity in $O(l^{h-1}p^2)$ [18] and optimal reinsertion (OR) neighborhood is in $O(2^k p^{k+1})$ [21]. In particular, LAGD complexity does not benefit from sparsity, $l^{h-1}$ being constant, whereas our approach is faster when $k$ decreases. Moreover computing the score difference of two DAGs before and after a SWAP operation is easy to do as it remains local to a single target node thanks to score decomposition.

Another source of suboptimality comes from the global acyclicity constraint of Bayesian networks.

## 3.2 Breaking cycles by successive deletions and swaps

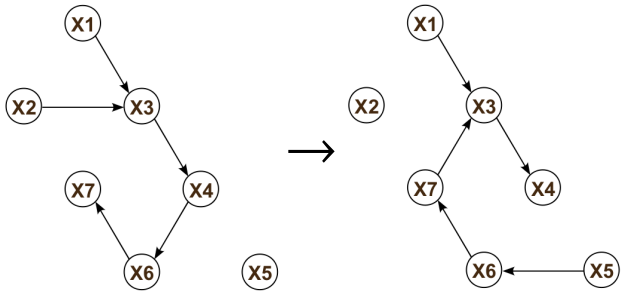


**Figure 4.** Applying an extended SWAP* operation breaking a cycle by an additional SWAP operation:
SWAP*($X2|X7 \rightarrow X3$) = {SWAP($X2|X7 \rightarrow X3$), SWAP($X4|X5 \rightarrow X6$)}.

Consider the 7-variable DAG example in Figure 4. Swapping the parent $X2$ of $X3$ by $X7$ in DAG $G$ (Fig. 4.left) introduces a directed cycle $\{X7 \rightarrow X3, X3 \rightarrow X4, X4 \rightarrow X6, X6 \rightarrow X7\}$ and is therefore forbidden in our $\mathcal{N}^{ADRS}$ neighborhood. However it may correspond to a large *local* score improvement with respect to variable $X3$. Let us denote this improvement by $\Delta_{X3}(G, \text{SWAP}(X2|X7 \rightarrow X3)) = f_{X3}(D, G') - f_{X3}(D, G)$ with $G'$ obtained by applying the SWAP operation on $G$ ($G'$ is not a valid DAG), and $D$ and $f$ being the sample and scoring function. Our idea is to heuristically guide the search for a second (or more) local operator to be applied on $G'$ in order to restore graph acyclicity ($G'$ becomes valid) and such that the *true* score of the final DAG is greater than the score of the original one. In Figure 4, it is obtained by applying a second SWAP.

For that purpose, we define an extended SWAP operator, denoted SWAP*, able to break all directed cycles by performing a succession

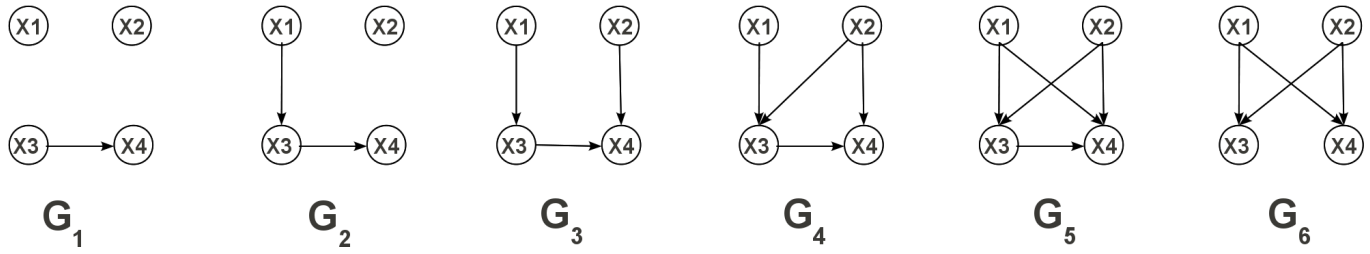


**Figure 3.** Problems with the classical neighborhood $\mathcal{N}^{ADR}$ when two nodes (here $X3$ and $X4$) have the same parents. True DAG is $G_6$. Starting from the empty graph (not shown), $G_4$ can be locally optimum for $\mathcal{N}^{ADR}$ when the sample size is small (score of $G_5$ lower than $G_4$), whereas in $\mathcal{N}^{ADRS}$, SWAP($X3|X1 \rightarrow X4$) moves directly from $G_4$ to $G_6$.

**Algorithm 2**: SWAP*($X|Y \rightarrow Z$) operator.

**Input** : operation $X|Y \rightarrow Z$, sample $D$, score $f$, DAG $G(\mathbf{X}, \mathbf{E})$
**Output**: a set of local operations $\mathbf{L}$
$\mathbf{L} \leftarrow \emptyset$ /* *Initialize output operations to the empty set* */ ;
$\mathbf{X'} \leftarrow \mathbf{X}$ /* *Candidate parent set for future swaps* */ ;
$G' \leftarrow G$ /* *Copy of input DAG* */ ;
3 $\Delta = \Delta_Z(G', \text{SWAP}(X|Y \rightarrow Z))$ /* *Putative score improvement* */ ;
**if** $\Delta > 0$ **then**
  $\mathbf{L} \leftarrow \mathbf{L} \cup \{\text{SWAP}(X|Y \rightarrow Z)\}$ ;
  Apply SWAP($X|Y \rightarrow Z$) to $G'$ ;
  /* *Repeat deletion or swap operations until no more cycles*
4  */ **while** $\Delta > 0 \wedge (\boldsymbol{C} \leftarrow$`NextCycle`$(G')) \neq \emptyset$ **do**
5  $\mathbf{X'} \leftarrow \mathbf{X'} \setminus nodes(\mathbf{C})$ ;
  /* *Choose the best deletion to break cycle* $\boldsymbol{C}$ */ ;
6  $(U^* \rightarrow W^*) \leftarrow$
  $\text{argmax}_{(U \rightarrow W) \in \mathbf{C} \setminus \{Y \rightarrow Z\}} \Delta_W(G', \text{DELETE}(U \rightarrow W))$ ;
  /* *Test if the sum of local score changes is positive* */ ;
  **if** $\Delta + \Delta_{W^*}(G', DELETE(U^* \rightarrow W^*)) > 0$ **then**
    $\mathbf{L} \leftarrow \mathbf{L} \cup \{\text{DELETE}(U^* \rightarrow W^*)\}$ ;
    $\Delta \leftarrow \Delta + \Delta_{W^*}(G', \text{DELETE}(U^* \rightarrow W^*))$ ;
    Apply DELETE($U^* \rightarrow W^*$) to $G'$ ;
  **else**
    /* *Choose the best swap to get a positive change* */ ;
7  $(U^*|V^* \rightarrow W^*) \leftarrow$
    $\text{argmax}_{(U \rightarrow W) \in \mathbf{C}, V \in \mathbf{X'}} \Delta_W(G', \text{SWAP}(U|V \rightarrow W))$ ;
    $\Delta \leftarrow \Delta + \Delta_{W^*}(G', \text{SWAP}(U^*|V^* \rightarrow W^*))$ ;
    **if** $\Delta > 0$ **then**
      $\mathbf{L} \leftarrow \mathbf{L} \cup \{\text{SWAP}(U^*|V^* \rightarrow W^*)\}$ ;
      Apply SWAP($U^*|V^* \rightarrow W^*$) to $G'$ ;
    **else**
8  $\mathbf{L} \leftarrow \emptyset$ /* *Abort all local operations* */ ;
**return** $\mathbf{L}$ ;

of deletion or swap operations. It can be seen as a kind of greedy descent search in the space of directed cyclic graphs, trying to remove the less important arcs or to swap them in order to compensate for their loss, until a better valid DAG is found. We use local score changes to guide the search: $\Delta_{Xi}(G, OP) = f_{Xi}(D, G') - f_{Xi}(D, G)$, with $G'$ the result of applying the local move operator $OP \in \{DELETE, SWAP\}$ to $G$. A negative sum of local changes aborts the search. Recall that finding a minimum number of arc deletions in order to restore acyclicity is NP-hard, see *minimum feedback arc set problem* in [11]. We use a greedy approach instead. The pseudo-code of SWAP* is given in Algorithm 2. The local score improvement of the initial SWAP operation is evaluated at line 3. It corresponds to a putative gain on the current score. If it is positive then this operation is applied to a copy of the input DAG $G$, checking next if it creates some directed cycles. Each cycle is retrieved by the `NextCycle` function[7] and the algorithm searches for an arc deletion in this cycle with minimum deterioration of the local score at line 6. If the combined local score change of the SWAP and DELETE operations is positive then it applies the selected arc deletion and continues to test if there are no more directed cycles at line 4. If the combined local score change is negative then it tries to swap an arc of the cycle such that the combined local score change is maximized (line 7) and positive. If it fails to find such an operation then it stops breaking cycles and returns an empty operation set. Finally if it succeeds, breaking all cycles, then it returns a feasible set of SWAP and DELETE operations resulting into a new valid DAG $G'$ with a better score than $G$. The true score improvement is equal to $\Delta$.

Algorithm 2 terminates because there are $O(pk)$ directed cycles to break, assuming $p$ nodes and $k$ maximum number of parents per node. And new arcs created by swap operations cannot be swapped again more than $p$ times thanks to our restricted list of alternative candidate parent nodes $\mathbf{X'}$ used at line 7, initialized to all the variables at the beginning, and updated by the list of nodes present in the current cycle at line 5.

We apply the same approach to extend the other operators ADD* and REVERSE*, breaking cycles with deletions or swaps. Because REVERSE*($X \rightarrow Y$) = ADD*($Y \rightarrow X$), we use only ADD*. The resulting neighborhood exploiting these extended operators is denoted $\mathcal{N}^{A^*DS^*}$ and SGS using this neighborhood is denoted SGS[3] (Stochastic Greedy Search with Successive Swaps) in the experiments.

## 4 Experimental Results

In this section, we describe a set of experiments aimed at testing the performance of SGS$^i$ algorithms compared with state-of-the-art Bayesian network structure learning algorithms on standard Bayesian networks and challenging gene regulatory networks.

### 4.1 Results on Standard Bayesian Networks

We used four gold-standard networks from the Bayesian Network Repository[8] whose main properties are shown in Table 1. In this table, *Nodes* and *Arcs* specify the number of nodes and arcs respectively in the DAG. The *Max in-degree* is the maximum number of parents of a node. The *Min-max states* are the minimum and maximum number of states in the domain of a variable associated to a node. Finally, *Longest path* is the length of the longest directed path

[7] We implemented an incremental acyclicity test [15] which returns a shortest directed cycle using breadth first search.
[8] `http://www.cs.huji.ac.il/site//labs/compbio/Repository/`

between any pair of nodes in the DAG. 100 samples of size $n = 500$ and $n = 5000$ were generated for each network using Causal Explorer [1].

**Table 1.** Bayesian network properties

| | Alarm | Insurance | Hailfinder | Pigs |
|---|---|---|---|---|
| Nodes | 37 | 27 | 56 | 441 |
| Arcs | 46 | 52 | 66 | 592 |
| Max in-degree | 4 | 3 | 4 | 2 |
| Min-max states | 2-4 | 2-5 | 2-11 | 3-3 |
| Longest path | 11 | 10 | 14 | 6 |

We compared SGS$^i$ algorithms with LAGD [18], available in the WEKA software [16] and GES [4] implemented in Tetrad 4.4.0 [25]. LAGD was shown to outperform repeated hill-climbing, order-based search, tabu search, and simulated annealing in [24]. GES, which is considered the reference algorithm for Bayesian network structure learning, was reported having similar performance to the most recent order-based search algorithms in [2]. Recall that SGS$^1$ is similar to repeated hill-climbing, SGS$^2$ uses the SWAP operator, and SGS$^3$ breaks cycles by successive DELETE and SWAP operators. Experiments were performed on a 3 GHz Intel Core2 computer with 4GB running Linux 2.6.

We fixed the maximum number of parents per node at $k = 5$ for SGS$^i$ and LAGD. LAGD exploits a $h = 2$-look ahead in $l = 5$ good directions. GES was restricted on the number of adjacent nodes: $d = 7$ for Hailfinder and $d = 10$ for Pigs network as done in [2]. All the methods started from an empty graph and optimized the BDeu score with *equivalent sample size* $\alpha = 1$ and no prior on the network structures. For each sample, we recorded the best score obtained by GES, and by $r = 10$ randomized greedy searches for SGS$^i$ (Algorithm 1 at line 2) as for LAGD[9].

In order to test the statistical significance of the difference in BDeu score between two methods, we applied a non-parametric paired test, the Wilcoxon signed-rank test [33]. Table 2 presents the test results for all the pairs of methods by using an unilateral alternative (no difference versus better) and a familywise error rate of 5% that gives by Bonferroni correction for multiple comparisons a pairwise type I error of $6.2510^{-4}$. The above results were summarized in Table 3, in a summarized Wilcoxon score for each method, obtained by summing the positive comparisons and subtracting the negative ones. The non significant comparisons were ignored.

SGS$^3$ was the best method for the four networks, except for Pigs network with $n = 5000$ which is better learned by GES. We conjecture that in this case, GES was closed to its asymptotic optimal behavior, which may be due to the smaller in-degree, domain size, and longest path of Pigs network compared to the other networks. SGS$^2$ improved on SGS$^1$ and reached the second position especially for small sample sizes for the reasons shown in Figure 3. LAGD got poor results, the difference with SGS$^1$ can be explained by a better randomization in SGS$^1$ (Algorithm 1 at line 1). LAGD failed on the Pigs network due to the large number of variables $p = 441$ that makes the exploration of 2-look ahead neighborhoods infeasible in a reasonable time. GES was the worst method of this evaluation (except for Pigs) due to the small sample sizes we used.

Although the algorithms are designed to maximize a (BDeu) score, we generally look for a network structure as close as possible to the

[9] We randomly permute the input variables at each run.

**Table 2.** Wilcoxon test comparing pairs of algorithms (familywise error rate = 5%). For Method1 versus Method2, + means that Method1 is significantly better than Method2, − means that Method1 is significantly worst than Method2 and ∼ means there is no significant result

| | Alarm | | Insurance | |
|---|---|---|---|---|
| Sample size | 500 | 5k | 500 | 5k |
| SGS$^3$ vs SGS$^1$ | + | + | + | + |
| SGS$^3$ vs SGS$^2$ | + | + | + | + |
| SGS$^2$ vs SGS$^1$ | + | ∼ | ∼ | ∼ |
| SGS$^3$ vs GES | + | + | + | + |
| SGS$^3$ vs LAGD | + | + | + | + |
| SGS$^2$ vs GES | + | ∼ | + | + |
| SGS$^2$ vs LAGD | ∼ | ∼ | + | + |
| SGS$^1$ vs GES | + | ∼ | + | + |
| SGS$^1$ vs LAGD | ∼ | ∼ | + | + |
| LAGD vs GES | + | ∼ | + | + |
| | Hailfinder | | Pigs | |
| SGS$^3$ vs SGS$^1$ | ∼ | + | + | + |
| SGS$^3$ vs SGS$^2$ | ∼ | + | + | + |
| SGS$^2$ vs SGS$^1$ | ∼ | ∼ | ∼ | ∼ |
| SGS$^3$ vs GES | + | + | + | - |
| SGS$^3$ vs LAGD | ∼ | + | n/a | n/a |
| SGS$^2$ vs GES | + | + | ∼ | - |
| SGS$^2$ vs LAGD | ∼ | + | n/a | n/a |
| SGS$^1$ vs GES | + | + | ∼ | - |
| SGS$^1$ vs LAGD | ∼ | + | n/a | n/a |
| LAGD vs GES | + | + | n/a | n/a |

**Table 3.** Summarized Wilcoxon scores

| | Alarm | Insurance | Hailfinder | Pigs |
|---|---|---|---|---|
| SGS$^3$ | 8 | 8 | 5 | 4 |
| SGS$^2$ | 0 | 2 | 2 | -3 |
| SGS$^1$ | -2 | 2 | 2 | -3 |
| LAGD | -1 | -4 | -1 | n/a |
| GES | -5 | -8 | -8 | 2 |

**Table 4.** Spurious edges (+) and missing edges (-) to sum for the structural Hamming distance. Both scores are in bold when giving the best distance per configuration

| | Alarm | | Insurance | |
|---|---|---|---|---|
| Sample size | 500 | 5k | 500 | 5k |
| SGS$^3$+ | **8** | 6 | **4** | **2** |
| SGS$^3$- | **3** | 2 | **20** | **8** |
| LAGD+ | 11 | 8 | **4** | 5 |
| LAGD- | 4 | 2 | **20** | 11 |
| GES+ | **6** | **4** | 2 | 3 |
| GES- | **5** | **2** | 23 | 12 |
| | Hailfinder | | Pigs | |
| SGS$^3$+ | 17 | **16** | 32 | 41 |
| SGS$^3$- | 24 | **13** | 0 | 0 |
| LAGD+ | 21 | 20 | n/a | n/a |
| LAGD- | 26 | 19 | n/a | n/a |
| GES+ | **15** | 11 | **2** | **0** |
| GES- | **24** | 22 | **7** | **0** |

true one. When the sample size grows large, improving the BDeu score leads to reduce the *structural Hamming distance* (SHD) between the learned and the true network. We report in Table 4 the means over 100 datasets (rounded values to the nearest integer) of the missing and spurious edges without taking into account the edge orientations. The SHD is the sum of the above values. We do not report SGS$^1$ and SGS$^2$ results that are outperformed by SGS$^3$. SGS$^3$ (resp. GES) got the best SHD in 4/8 (resp. 5/8) configurations and outperformed LAGD (which won in 1/6). GES performed extremely well on the Pigs network, finding the true network with 5,000 samples, whereas SGS$^3$ learned too many edges but recovered all the true edges (even with $n = 500$). The spurious edges learned by SGS$^3$ are exclusively due to random orientations of compelled arcs in v-structures (see Figure 1). Assuming $X1 \rightarrow X3 \leftarrow X2$ in the true network (v-structures are very frequent in the Pigs network) and a large sample size, if during its greedy search SGS$^3$ adds first $X1 \leftarrow X3$ and $X3 \rightarrow X2$ then it will add next a *covering edge* $X1 \rightarrow X2$ or $X1 \leftarrow X2$ in order to find a minimal independence map (see Proposition 1). However the true v-structure can be regain by comparing for each covered v-structure the 3 possible non-covered v-structures independently from the other variables and selecting the highest score configuration. After this post-processing step on Pigs network, we had only 1 (resp. 2) spurious edges for SGS$^3$ with 500 (resp. 5000) samples without increasing the missing edges number. The same post-processing had no effect on the other smaller networks.

## 4.2 Detailed analysis on the Alarm network

We conducted a series of algorithm analyzes on the Alarm network with a sample size $n = 500$.

**Table 5.** Single greedy search analysis on the Alarm network ($n = 500, r = 1$)

| | BDeu score | Iter. | BDeu cache | Time(s) |
|---|---|---|---|---|
| SGS$^3$ | -5490.53 | 66 | 4543 | 3.2 |
| SGS$^2$ | -5513.89 | 58 | 4310 | 2.3 |
| SGS$^1$ | -5541.55 | 55 | 3305 | 1.5 |
| LAGD | -5544.61 | 35 | 4782 | 3.2 |
| GES | -5659.26 | 72 | 5531 | 2.4 |

Table 5 shows a typical example of the BDeu score reached by the algorithms for a single greedy search ($r = 1$) on a particular sample. It also reports the number of local moves *Iter.* (Algorithm 1 at line 1), the number of local score $f_{Xi}(D, G)$ computations for *different* parent configurations, *i.e.* assuming a *perfect BDeu cache*, and the CPU time in seconds. As expected, the number of iterations for LAGD was less than half of the other methods, since LAGD applies two local operators (ADD, DELETE, or REVERSE) at each move. The swap operations used by SGS$^2$ and SGS$^3$ slightly increased the number of moves allowing to find better local optima. Comparing CPU times is a difficult task due to the various implementation details (language choice, caching strategy, sample compilation techniques, see for instance Chapter 18.4.3 in [19]). Nevertheless SGS$^i$ algorithms are comparable. In our implementation, SGS$^3$ was two times slower than SGS$^1$. A more fair comparison is the number of calls to new local score computations. SGS$^1$ needed the least number, whereas GES needed the most as it explores a larger neighborhood in the space of Markov-equivalent networks. Notice that SGS$^2$ and SGS$^3$ required less score computations than LAGD due to graph sparsity (see Section 3.1). The greater BDeu cache size of SGS$^3$ compared to SGS$^2$ is mainly due to the increased number of local moves.

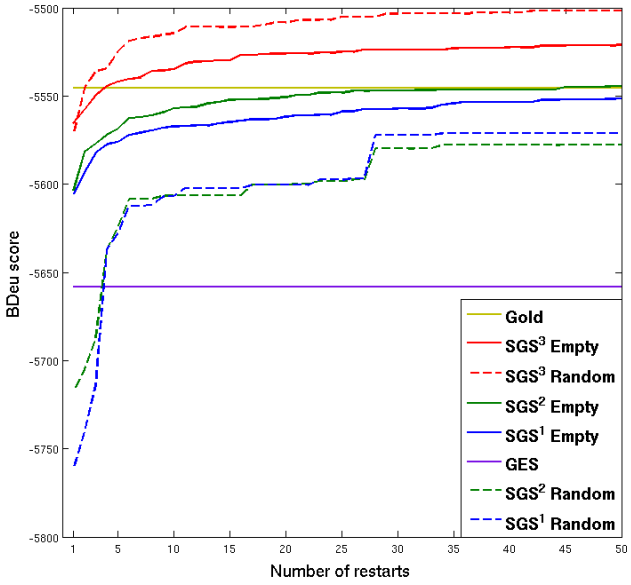


**Figure 5.** Best BDeu scores, averaged on 30 Alarm samples ($n = 500$), found by SGS$^i$ algorithms as the number of restarts $r$ increases and starting either from an empty (solid line) or a random graph (dashed line). Results of GES (bottom line) and BDeu score of the true network *Gold* (top line) are also given. Methods are sorted by decreasing BDeu score at $r = 1$

We further analyzed the impact on performances of the number of restarts $r$ and the initial graph used by SGS$^i$ algorithms (see `InitGraph` in Algorithm 1). Figure 5 reports averaged BDeu scores on 30 Alarm samples (the length of the 95% confidence interval of the BDeu score mean was around ±30). Methods are sorted from the best, SGS$^3$ with an empty graph, to the worst, SGS$^1$ with a random initial graph, at $r = 1$. Initial random graphs are composed of 71 arcs with at most two parents per node[10]. All SGS$^i$ methods reached a better BDeu score than GES in this small sample size situation by the use of less than $r = 4$ restarts. SGS$^i$ methods converged quite rapidly as $r$ increases. Only SGS$^3$ found a better score than the true network. Initial random graphs were counter-productive for all the methods, except for SGS$^3$. It shows that starting from a random graph is useful if the available operators allow to move efficiently in the search space, which is illustrated by the following experiment. On the contrary, using randomness within the greedy selection process (see `SelectRandom` in Algorithm 1) was always beneficial.

Finally, we analyzed how often our new local move operators are used during the search. Table 6 shows the mean number of local moves using a specific operator averaged on $r = 50$ greedy searches over 30 Alarm samples ($n = 500$). Here, ADD+ and SWAP+ mean that at least two local operators were successively applied at a given move in order to break cycles[11]. The new local move operators are

[10] For each node, we randomly select two parents and remove a parent if it creates a directed cycle.

[11] We found at most 4 (resp. 5) successive operations for the 1500 runs starting from an empty (resp. random) graph.

**Table 6.** Operator usage depending on the initial graph

| | Empty graph init. | | | Random graph init. | | |
|---|---|---|---|---|---|---|
| | $SGS^1$ | $SGS^2$ | $SGS^3$ | $SGS^1$ | $SGS^2$ | $SGS^3$ |
| ADD | 53 | 53 | 52 | 53 | 9.5 | 11.5 |
| DELETE | 0.5 | 0.1 | 1.4 | 64 | 20 | 43 |
| REVERSE | 0.9 | 1.1 | 0.7 | 5.3 | 3.4 | 1.9 |
| SWAP | 0 | 1.1 | 3.2 | 0 | 47 | 33 |
| ADD+ | 0 | 0 | 2 | 0 | 0 | 14 |
| SWAP+ | 0 | 0 | 1.6 | 0 | 0 | 9.5 |

mostly used when starting from a random graph (there are more cycles to break), but when starting from an empty graph, only a few applications of the extended moves allow to significantly improve the BDeu score as previously shown in Figure 5 and Table 3.

## 4.3 Results on Gene Regulatory Networks

Gene regulatory network reconstruction from gene expression data using Bayesian network structure learning was first proposed in [12]. We used simulated expression datasets of the DREAM5 Systems Genetics Challenge A [10]. Genetics data were not used as they require additional modelling to be taken into account, see *e.g.* [32]. Expression data were generated using the SysGenSIM generator [23] based on ordinary differential equation simulation. Five datasets are available for three different sample sizes ($n$ = 100, 300, and 999). The 15 datasets were obtained from *different* known gene networks composed of $1,000$ variables and containing directed cycles. For each sample size, the five network structures contain a different number of edges varying from $\approx 2,000$ (*Net1*) to more than $5,000$ (*Net5*). We discretized gene expression levels into 2 to 4 states using an adaptive method based on an adapted $k$-means algorithm and the more general framework of Gaussian mixture models as described in [32].

With such large networks, we had to adapt the learning procedure of $SGS^i$ algorithms[12]. We decided to restrict their lists of candidate parents as done in [14]: we selected for each variable $X$ the set of parents $\mathbf{S}$ such that each element $Y$ of $S$ improves the local BDeu score when it is considered as a unique parent compared to the orphan situation ($f_X(D, \{Y \rightarrow X\}) > f_X(D, \emptyset)$). This filtering process was done before the search[13]. In these experiments, $SGS^i$ algorithms have a maximum number of parents per node fixed at $k = 5$ and use $r = 10$ restarts. Instead of LAGD (which was too slow), we used MMHC [31] having two steps similar to $SGS^i$. It first selects the skeleton of the graph using mutual information measures (MMPC [30] filtering step) and then orientates edges in a greedy manner. We recorded the best BDeu score of 10 runs for MMHC, by randomly permuting the input variables at each run. The skeleton being built locally for each variable, MMHC can handle large networks. All the methods started from an empty graph and optimized the BDeu score with $\alpha = 1$ and no prior on the network structures.

It was difficult to perform statistics with the results of this experiment. Indeed, contrary to the standard network experiment, there were no replicated samples of the same network. We decided to pool results per sample size and we performed the Wilcoxon test on the groups. With only five results per group to compare the methods, we know that the power of the test is very low. So, we applied a pairwise type I error of 5% and we did not try to correct for multiple comparisons, see Table 7. However, it is worth to note that $SGS^3$ was always the best method and that it increased the BDeu score by about 2% in average.

[12] GES managed in ~1-hour CPU time each network thanks to its better implementation of caching and heap data structure.

[13] It could also be done during the search as in [13].

**Table 7.** Wilcoxon test (error rate = 5%) for different gene network sample sizes

| | 100 | 300 | 999 |
|---|---|---|---|
| $SGS^3$ vs MMHC | + | + | + |
| $SGS^3$ vs GES | + | + | + |
| MMHC vs GES | - | ~ | + |

Surprisingly, GES appeared to be better on smaller sample sizes compared to MMHC. As explained below, MMHC was penalized by its filtering process, especially on the smallest sample size, whereas GES had no restrictions on the candidate parent sets.

In these experiments, the structural Hamming distance (SHD) was not informative due to the poor results reached by all the algorithms for such large networks. SHD, equal to the sum of spurious edges ($s$) and missing edges ($m$), was greater than the number of edges ($e$) of the true graph. In this situation, even the empty structure appears better. For this reason, we computed another aggregated structural quality measure based on the *precision* ($\frac{e-m}{e-m+s}$) and *recall* ($\frac{e-m}{e}$) measures and took the Euclidean distance to the origin to combine them ($\sqrt{precision^2 + recall^2}$). Contrary to SHD, a high distance indicates a better structural quality. We observed in Table 8 contrasted performances between the tested methods depending on the sample size: for n=100, MMHC got the best results, for $n = 300$, it was GES, and finally $SGS^3$ performed the best for the largest sample size. Better BDeu scores are not always synonymous with a better structural quality, the limited sample size in addition to the non faithfulness of the data (generated by ordinary differential equations with cycles) could explain this behavior. We should notice that the good optimization behavior of $SGS^3$ in terms of BDeu scores resulted in a better structural quality than GES and MMHC as sample size increases.

**Table 8.** Euclidean distances to the origin of the (precision, recall) values. Best distances per sample size are in bold

| $n$ | | $SGS^3$ | MMHC | GES |
|---|---|---|---|---|
| 100 | *Net1* | 0.170 | **0.218** | 0.196 |
| | *Net2* | 0.213 | **0.295** | 0.232 |
| | *Net3* | 0.214 | **0.266** | 0.236 |
| | *Net4* | 0.201 | **0.265** | 0.214 |
| | *Net5* | 0.206 | **0.247** | 0.243 |
| 300 | *Net1* | **0.510** | 0.483 | 0.464 |
| | *Net2* | 0.342 | 0.337 | **0.385** |
| | *Net3* | 0.484 | 0.488 | **0.505** |
| | *Net4* | 0.453 | 0.478 | **0.498** |
| | *Net5* | 0.419 | 0.397 | **0.428** |
| 999 | *Net1* | **0.578** | 0.537 | 0.549 |
| | *Net2* | **0.581** | 0.510 | 0.505 |
| | *Net3* | 0.454 | 0.441 | **0.484** |
| | *Net4* | **0.476** | 0.450 | **0.476** |
| | *Net5* | **0.479** | 0.471 | 0.458 |

Moreover, we compared the quality of the filtering process used by MMPC and $SGS^3$. Table 9 reports for the first network (*Net1*) of each sample size, the total number of arcs kept by the filtering process and the recall value, which represents the percentage of true edges

in the filter. Our first observation is the poor recall of both filtering processes, which indicates strong structural differences between the true network and the learned network even with $n = 999$ sample size. Our filtering approach obtained better recall values with similar compression sizes than MMPC.

**Table 9.** Total size and recall of candidate parent lists for $SGS^3$ and MMPC on the most sparse gene network *Net1*

| | | 100 | 300 | 999 |
|---|---|---|---|---|
| BDeu test | size | 2670 | 2430 | 5984 |
| | recall | 9% | 18% | 35% |
| MMPC | size | 2568 | 3064 | 3842 |
| | recall | 5% | 12% | 23% |

Finally, we tried the same methods as in Section 4.1 on 50 small gene regulatory networks (*Web50* Artificial Gene Networks with 50 nodes and 50 arcs) without doing any filtering process. For a sample size $n = 500$, $SGS^3$ was significantly better than GES and LAGD in terms of BDeu scores and slightly better in terms of Euclidean distances to the origin of the (precision, recall) values.

## 5 Conclusion

We have presented in this paper a new greedy search algorithm called $SGS^i$ exploiting stochasticity from two random draws. We have developed a new local move operator called SWAP and extended versions for ADD and SWAP operators to overcome frequent limitations of local search methods which are local maxima and cyclic situations. We compared $SGS^3$ using SWAP and extended operators to state-of-the-art methods and we obtained significant BDeu and recall value improvements on classical benchmarks and also simulated gene regulatory network datasets when the sample size is small. The complexity of $SGS^3$ stays moderate with sparse networks. In case of large networks with many variables we applied a filtering process in preprocessing using the same criterion as for the search. This process kept more edges from the true network than mutual information-based methods with significant reduction of the search space.

In the future, we would like to test our new operators with other local search methods like tabu search.

## REFERENCES


[1] C. Aliferis, I. Tsamardinos, A. Statnikov, and L. Brown, 'Causal Explorer: A Probabilistic Network Learning Toolkit for Biomedical Discovery', in *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, (2003).

[2] J. Alonso-Barba, L. de la Ossa, and J. Puerta, 'Structural learning of bayesian networks using local algorithms based on the space of orderings', *Soft Comput*, 1881–1895, (2011).

[3] W. Buntine, 'Theory Refinement on Bayesian Networks', in *Proc. of UAI-91*, pp. 52–60, San Mateo, CA, (1991).

[4] D. Chickering, 'Optimal structure identification with greedy search', *Journal of Machine Learning Research*, **3**, 507–554, (2002).

[5] D. Chickering, D. Heckerman, and C. Meek, 'Large-Sample Learning of Bayesian Networks is NP-Hard', *Journal of Machine Learning Research*, **5**, 1287–1330, (2004).

[6] D. Chickering and D. Heckermann, 'Learning bayesian networks is NP-complete', *In learning from data: Al and Statistics*, (1996).

[7] G Cooper and E Hersovits, 'A Bayesian method for the induction of probabilistic networks from data', *Machine Learning*, **9**, 309–347, (1992).

[8] R. Daly and Q. Shen, 'Learning Bayesian Network Equivalence Classes with Ant Colony Optimization', *Journal of Artificial Intelligence Research*, **35**, 391–447, (2009).

[9] R. Daly, Q. Shen, and S. Aitken, 'Learning Bayesian networks: approaches and issues', *The Knowledge Engineering Review*, **26**(2), 99–157, (2011).

[10] 'The DREAM5 Systems Genetics Challenges'. http://wiki.c2b2.columbia.edu/dream/index.php/D5c3, 2010.

[11] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende, 'Feedback Set Problems', in *Encyclopedia of Optimization*, 1005–1016, Springer, (2009).

[12] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, 'Using bayesian networks to analyse expression data', *Journal of computational biology*, **7**(3/4), 601–620, (2000).

[13] J. Gámez, J. Mateo, and J. Puerta, 'Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood', *Data Min. Knowl. Discov.*, **22**, 106–148, (2011).

[14] A. Goldenberg and A. Moore, 'Tractable learning of large Bayes net structures from sparse data', in *Proc. of ICML'04*, pp. 44–51, (2004).

[15] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. Tarjan, 'Faster algorithms for incremental topological ordering', in *Proc. of ICALP*, pp. 421–433, (2008).

[16] M. Hall, F. Eibe, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, 'The WEKA Data Mining Software', *SIGKDD Explorations*, **11**, (2009).

[17] D. Heckerman, D. Geiger, and D. Chickering, 'Learning Bayesian Networks: The Combination of Knowledge and Statistical Data', in *Machine Learning*, volume 20, pp. 197–243, (1995).

[18] A. Holland, M. Fathi, M. Abramovici, and M. Neubach, 'Competing fusion for bayesian applications', in *Proc. of IPMU 2008*, pp. 378–385, (2008).

[19] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, 2009.

[20] G. Melançon, I. Dutour, and M. Bousquet-Mélou, 'Random generation of directed acyclic graphs', *Electronic Notes in Discrete Mathematics*, **10**, 202–207, (2001).

[21] A. Moore and W.K. Wong, 'Optimal reinsertion: A new search operator for accelerated and more accurate bayesian network structure learning', in *Proc. of ICML '03*, pp. 552–559, (2003).

[22] J. Nielsen, T. Kocka, and J. Pefia, 'On Local Optima in Learning Bayesian Networks', in *Proc. of UAI-03*, pp. 435–442, (2003).

[23] A. Pinna, N. Soranzo, I. Hoeschele, and A. de la Fuente, 'Simulating system genetics data with SysGenSIM', *Bioinformatics*, **27**, 2459–2462, (2011).

[24] E. Salehi and R. Gras, 'An empirical comparison of the efficiency of several local search heuristics algorithms for Bayesian network structure learning', in *Learning and Intelligent OptimizatioN Workshop (LION 3)*, (2009).

[25] R. Scheines, P. Spirtes, C. Glymour, C. Meek, and T. Richardson, 'The TETRAD Project: Constraint Based Aids to Causal Model Specification', *Multivariate Behavioral Research*, **33**(1), 65–117, (1998).

[26] M. Schmidt, A. Niculescu-Mizil, and K. Murphy, 'Learning graphical model structure using L1-regularization paths', in *Proc. of AAAI'07*, pp. 1278–1283, (2007).

[27] G. Schwarz, 'Estimating the dimension of a model', *The annals of statistics*, (1978).

[28] P. Spirtes, C. Glymour, and R. Scheines, 'Causality from probability', *Evolving knowledge in the natural and behavioral sciences*, 181–199, (1990).

[29] M. Teyssier and D. Koller, 'Ordering-based Search: A Simple and Effective Algorithm for Learning Bayesian Networks', in *Proc. of UAI'05*, pp. 584–590, (2005).

[30] I. Tsamardinos, C. Aliferis, and A. Statnikov, 'Time and sample efficient discovery of Markov blankets and direct causal relations', in *Proc. of KDD'03*, pp. 673–678, (2003).

[31] I. Tsamardinos, L. Brown, and C. Aliferis, 'The max-min hill-climbing Bayesian network structure learning algorithm', *Mach. Learn.*, **65**, 31–78, (2006).

[32] M. Vignes, J. Vandel, D. Allouche, N. Ramadan-Alban, C. Cierco-Ayrolles, T. Schiex, B. Mangin, and S. de Givry, 'Gene Regulatory Network Reconstruction Using Bayesian Networks, the Dantzig Selector, the Lasso and Their Meta-Analysis', *PLoS ONE*, **6**, (2011).

[33] F. Wilcoxon, 'Individual Comparisons by Ranking Methods', *Biometrics Bulletin*, **1**, 80–83, (1945).