Improved Local Search in Bayesian Networks Structure Learning

Mauro Scanagatta MAURO@IDSIA.CH
IDSIA [*] *, SUPSI* [†] *, USI* [‡] *- Lugano, Switzerland*
Giorgio Corani GIORGIO@IDSIA.CH
IDSIA [*] *, SUPSI* [†] *, USI* [‡] *- Lugano, Switzerland*
Marco Zaffalon ZAFFALON@IDSIA.CH
IDSIA [*] *- Lugano, Switzerland*

Abstract

We present a novel approach for score-based structure learning of Bayesian network, which couples an existing ordering-based algorithm for structure optimization with a novel operator for exploring the neighborhood of a given order in the space of the orderings. Our approach achieves state-of-the-art performances in data sets containing thousands of variables.

Keywords: Bayesian networks, Learning Graphical Models, Heuristic Search

1. Introduction

We consider the problem of learning the structure of a Bayesian network. We focus on score-based learning, namely finding the structure which maximizes a score that depends on the data; this task is NP-hard (Chickering et al., 2003).

Usually structural learning is accomplished in two steps: *parent set identification* and *structure optimization*. Parent set identification produces a list of suitable candidate parent sets for each variable. *Structure optimization* assigns a parent set to each node, maximizing the score of the resulting structure without introducing cycles.

Regarding the latter step, several exact algorithms have been developed based on dynamic programming (Koivisto and Sood, 2004; Silander and Myllymaki, 2006), branch and bound (de Campos et al., 2009; van Beek and Hoffmann, 2015), linear and integer programming (Cussens, 2011; Jaakkola et al., 2010), shortest-path heuristic (Yuan and Malone, 2012, 2013). These methods achieve good performance on smaller instances, but fail to scale to more than 100 variables unless the maximum in-degree (number of parents per node) is severely limited, greatly diminishing the expressiveness of the final network.

The need for a hard constraint on the maximum in-degree has been overcome by approximated techniques for exploring the parent sets (Scanagatta et al., 2015), which compute the score only of the most promising parent set for each node, without limits on the in-degree. By coupling approximated exploration of the parent sets and ordering-based structural learning, it has recently become possible to learn Bayesian networks even from *thousands*

∗. Istituto Dalle Molle di studi sull'Intelligenza Artificiale (IDSIA)
†. Scuola universitaria professionale della Svizzera italiana (SUPSI)
‡. Università della Svizzera italiana (USI)

of variables (Scanagatta et al., 2015; Lee and van Beek, 2017). The common idea of such approaches is to sample an order, identify the highest-scoring network given the order and eventually explore local changes to the order in order to further improve the score.

We consider different operators (Teyssier and Koller, 2005; Alonso-Barba et al., 2011) for improving the score of the structure by applying local changes to its underlying ordering. We then proposed the *window* operator for local search, already known as *block insertion* (Ben-Daya and Al-Fawzan, 1998) in the literature of local search. We couple the window operator with the ASOBS algorithm (Scanagatta et al., 2015), which learns a network structure given an ordering. The resulting algorithm consistently yields higher-scoring networks than ASOBS coupled with other operators, and the algorithms proposed by Lee and van Beek (2017).

All the software and data sets used in the experiments are available at `http://blip.idsia.ch`.

2. Structure Learning of Bayesian Networks

We consider the problem of learning the structure of a Bayesian Network from a complete data set of N instances $\mathcal{D} = \{D_1, ..., D_N\}$. The set of n categorical random variables is $\mathcal{X} = \{X_1, ..., X_n\}$. The structure of the network is defined by a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the collection of nodes and $\mathcal{E}$ is the collection of arcs. $\mathcal{E}$ can be defined as the set of parents $\Pi_1, ..., \Pi_n$ of each variable.

The goal of structure learning is to find a directed acyclic graph (DAG) $\mathcal{G}$ that maximizes a given score function, that is, we look for $\mathcal{G}_* = \text{argmax}_{\mathcal{G} \in \mathbf{G}}\, s(\mathcal{G}, D)$, where $\mathbf{G}$ is the set of all possible DAGs and s denotes the score function.

In this paper we adopt the BIC as a score function; asymptotically it is proportional to the posterior probability of the DAG. The BIC score is *decomposable*, namely it is constituted by the sum of the scores of the individual variables:

$$\begin{aligned} &\text{BIC}(\mathcal{G}) = \\ &= \sum\nolimits_{i=1}^{n} \text{BIC}(X_i, \Pi_i) = \sum\nolimits_{i=1}^{n} \sum\nolimits_{\pi \in |\Pi_i|} \sum\nolimits_{x \in |X_i|} \left(N_{x,\pi} \log \hat{\theta}_{x|\pi} \right) - \frac{\log N}{2} (|X_i| - 1)(|\Pi_i|)\,, \end{aligned}$$

where $\hat{\theta}_{x|\pi}$ is the maximum likelihood estimate of the conditional probability $P(X_i = x|\Pi_i = \pi)$, and $N_{x,\pi}$ represents the number of times $(X = x \wedge \Pi_i = \pi)$ appears in the data set, and $|\cdot|$ indicates the size of the Cartesian product space of the variables given as arguments (instead of the number of variables) such that $|X_i|$ is the number of states of X_i and $|\varnothing| = 1$.

Note however that the approach presented in this paper works for every decomposable scoring function, not only for the BIC.

Exploiting decomposability, it is possible to first identify independently for each variable X_i a list of candidate parent sets, denoted by C_i. We refer to this list as the *cache* of parent sets for X_i. The computation of the caches can be sped up by adopting exact pruning approaches (de Campos and Ji, 2011) and coupling them with approximated techniques for the exploration of the space of the parent sets (Scanagatta et al., 2015).

A variable X_j is a *potential parent* for X_i if it appears at least in one of the parent sets in C_i. We additionally compute for each variable X_i the set of its *potential parents* and we denote it by $\mathcal{P}_i$.

2.1 Ordering-Based Search (OBS)

Ordering-based search (OBS) has been proposed in (Teyssier and Koller, 2005). Given an ordering over the variables, the optimal network with respect to that ordering can be found in time $O(Ck)$, where $C = \sum_{i=1}^{n} |C_i|$ and k denotes the maximum in-degree.

For a given ordering $\prec$ the optimal parent set for the node X_i is:

$$Pa_{\prec}(X_i) = \arg \max_{\mathbf{U} \in U_{i,}} score(X_i, \mathbf{U}), \tag{1}$$

where $U_{i,}$ is the set of all the feasible parent sets, which contains only variables that precede X_i in the ordering $\prec$. We call *consistency rule* this constraint.

Once an ordering has been sampled, the highest-scoring structure given the order is quickly obtained following Eqn (1).

2.2 Acylic Selection OBS (ASOBS)

Acylic Selection OBS (ASOBS) has been proposed in (Scanagatta et al., 2015). It relaxes the consistency rule by allowing the introduction of back-arcs with respect to the original ordering, as long as they do not introduce a cycle. In particular, for each X_i it chooses the best parent set Π_i that does not create cycles with the already chosen parent sets Π_j for every $X_j \prec X_i$.

Since ASOBS generalizes OBS, for any given ordering $\prec$ it obtains a higher-scoring (equally-scoring in the worst case) network than OBS. In (Scanagatta et al., 2015), ASOBS is implemented without any operator for local search in the space of the orderings. This is instead the focus of this paper.

3. Operators for Local Search in the Space of the Orderings

Once we have found the highest-scoring network given an initial ordering (using either OBS or ASOBS), we look for local changes in the order that allow to further increase the score of the network. More precisely, we perform a local greedy hill-climbing search on the space of the orderings $\mathcal{O}$. In particular, an *operator* receives an ordering and returns a set of successor orderings. We discuss different operators in the following.

The *swap* operator is defined as:

$$Swap(i) : (X_1^{\prec}, ..., X_i^{\prec}, X_{i+1}^{\prec}, ..., X_n^{\prec}) \mapsto (X_1^{\prec}, ..., X_{i+1}^{\prec}, X_i^{\prec}, ..., X_n^{\prec}) \tag{2}$$

An example of swap is shown in Figure 1.

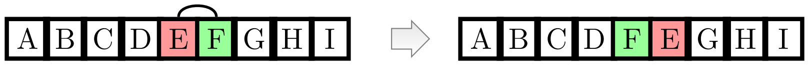

Figure 1: Example of operator *Swap*(5): the variables F and E are swapped in the ordering.

In order to compute the score of the highest-scoring network given the new order we need to be recompute (Teyssier and Koller, 2005) only the parent sets for $X_i^{\prec}$ and $X_{i+1}^{\prec}$ and $X_{i+1}^{\prec}$. Further computation can be saved by noting that:

1. the parent set of $X_{i+1}^{\prec}$ needs to be recomputed only if it contains $X_i^{\prec}$;
2. the parent set of $X_i^{\prec}$ needs to be recomputed only if $X_{i+1}^{\prec}$ is a *potential parent* for it: $\exists\, C^* \in C_{X_i^{\prec}}$ such that $X_{i+1}^{\prec} \in C^*$.

Using the *swap* operator, each ordering has $n-1$ candidate successors; thus one evaluates all such successors and eventually selects the one yielding the highest-scoring network.

A limit of the *swap* operator is its restricted neighborhood space, which limits its ability to escape local maxima. A more powerful operator, called *insertion*, was proposed in (Alonso-Barba et al., 2011). Given two indexes i and j, the variable at position i in the initial ordering is inserted at position j in the successor ordering, shifting the variables in the middle:

$$insertion(i,j) : (X_1^{\prec}, ..., X_j^{\prec}, ..., X_i^{\prec}, ..., X_n^{\prec}) \mapsto (X_1^{\prec}, ..., X_i^{\prec}, X_j^{\prec}, ..., X_n^{\prec}) \qquad (3)$$

Note that j can be lower or higher than i. An example of insertion is shown in Figure 2.

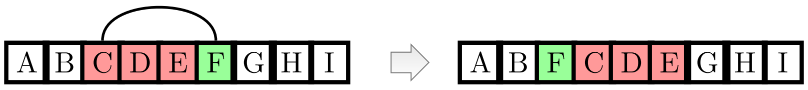

Figure 2: Example of *insertion*$(6,3)$: the variable F is inserted in the position occupied by variable C in the initial ordering.

The *swap* operator is thus a particular case of the *insertion* operator, in which it is applied to two adjacent nodes. Namely, *swap*(i)=*insertion(j+1,j)*.

Suppose we have three variables, $\{A, B, C\}$. The caches of parent sets for each variable are as follows:

A	B	C
$\{C\}$: -8	$\{A, C\}$: -7	$\{B\}$: -12
$\{\}$: -13	$\{A\}$: -9	$\{\}$: -16

Suppose we start from the ordering $\{A, B, C\}$, with optimal score -34. The successor orderings yielded by *swap* are $\{B, A, C\}$ (with optimal score -36) and $\{A, C, B\}$ (with optimal score -36). Instead, *insertion* yields also the order $\{C, A, B\}$, with optimal score -31 (this is the result of *insertion*$(3,1)$). Thus, given the same initial ordering, insertion explores a wider neighborhood than swap and for this reason it ends up with a higher-scoring solution. The drawback is a higher computational time spent given the same initial ordering, which allows insertion to explore less initial orderings than swap. We will show in Section 5 that the trade-off is however in favor of insertion, which generally yields a higher-scoring network than swap when both operators are provided with the same amount of computational time.

The insertion operator can be regarded as the result of multiple swaps applied to adjacent nodes (Figure 3). Each application of the swap operator requires computing the parent set for up to two variables. This results in the need to recompute only a limited number of parent sets also for insertion.

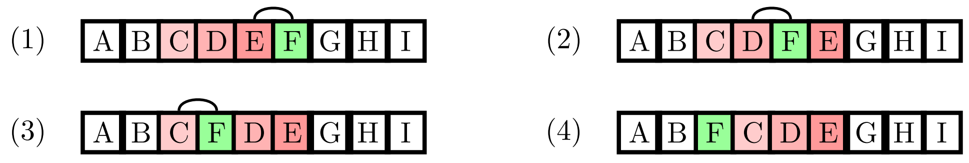

Figure 3: *Insertion* operator performed as a series of swap operation.

The *insertion* yields, given an ordering, $n \cdot (n-1)$ candidate successor orderings. In (Lee and van Beek, 2017; Alonso-Barba et al., 2011) the authors note that a complete assessment of the successors is cumbersome. They propose a new selection approach, named *hybrid*: a randomly chosen variable is used as a fixed index, and a complete exploration on the second index is then performed. This is executed by a series of single swap operation. The successor ordering with the highest score is then chosen. The search stops when no further improvement is found considering all the variables.

We now propose our operator, defined as $window(i, j, w)$, which generalizes the *insertion* operator. As already pointed out, this operator is known in the area of local search under the name of *block insertion* Ben-Daya and Al-Fawzan (1998). It works as follows. It selects the variables at position i in the original ordering and the $w - 1$ following variables. Such variables are inserted at position j in the new order; see the example in Figure 4.

$$window(i, j, w) :$$
$$(X_1^{\prec}, ..., X_j^{\prec}, ..., X_i^{\prec}, ..., X_{i+w}^{\prec}, ..., X_n^{\prec}) \mapsto (X_1^{\prec}, ..., X_i^{\prec}, ..., X_{i+w}^{\prec}, X_j^{\prec}, ..., X_n^{\prec})$$

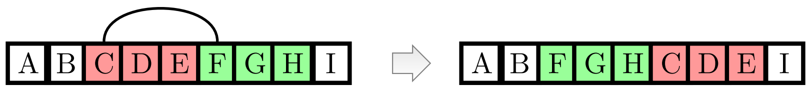

Figure 4: Example of $window(6, 3, 3)$: the variable F,G,H are inserted in the position previously occupied by C,D,E.

The *insertion* operator is thus a particular case of the window operator, characterized by $w = 1$. The window operator yields $(n - (w - 1)) \cdot (n - w)$ candidate successors for a given ordering.

The window operator can be equivalently regarded as a series of "window swap" operation, where each window swap requires recomputing the best parent set compatible with the ordering only for up to $w + 1$ variables. In particular we need to recompute the optimal parent set consistent with the ordering:

- For each of the w variables in the window $\{X_i^{\prec}, ..., X_{i+w}^{\prec}\}$, only if their current assigned parent set contains $X_{i-1}^{\prec}$;
- For $X_{i-1}^{\prec}$, only if any of the w variables in the window is a *potential parent* for it.

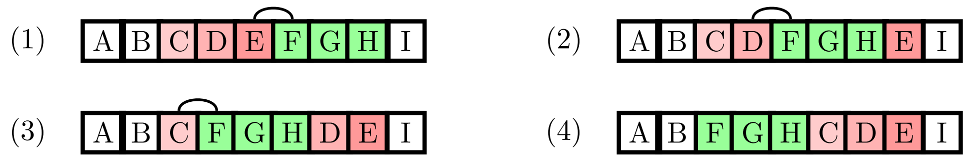

Figure 5: The *window* is equivalent to a series of window swaps.

The *window* operator can be applied to an ordering with the following schema, starting from $w = 1$.

- As the *hybrid* selection approach, a random variable is chosen as a fixed index. Consider all the successor states with window size w.
- If a successor state with improving score has been found, move to that state and set $w = 1$.
- If all the variables have been chosen and no successor state with improving score has been found, increase w by one. If it exceeds the maximum window size, return the current solution.

The window size w affects both the ability of the operator in escaping local maxima but also the required time for the computation, given a starting order. We experimentally found that the value w=5 is a good default choice.

4. Exploration meta-heuristic

Lee and van Beek (2017) suggest to enhance the performance of the greedy local search by applying a meta-heuristic. They proposed two approaches:

- IINOBS, based on iterated local search. It adopts the *insertion* operator coupled with a perturbation factor. The current local maxima is perturbed with random swaps until a specified termination condition is met.
- MINOBS, based on genetic algorithms. Individuals are generated with the *insertion* operator, and are later subject of a series of crossover and mutation stages.

We expand the idea of the iterated local search, adapting it in a way that it employs the *window insertion* operator with increasing window size. We call WINASOBS the resulting approach.

The algorithm starts with maximum window size $w = 1$. The parameters are listed in Table 1. At each iteration the following steps are executed:

1. A new ordering is sampled, and the starting solution is found through ASOBS.
2. Since ASOBS allows for back-arcs, the learned network can be incoherent with the given ordering. We thus extract a topological ordering consistent with the network learned by ASOBS; this is referred to as the *initial ordering*.

Parameter name	Description	Default value
p_a	number of swaps in the perturbation operator	3
p_b	max number of non-improving perturbations	10
p_c	number of iterations between maximum window size increase	5

Table 1: Parameters of WINASOBS

3. The initial ordering is improved using the *window insertion* operator and the *hybrid* selection, using the maximum window size w.
4. A perturbation factor based on a fixed number ($\frac{p_a}{100} \cdot n$) of random swaps is applied to the ordering, and the *window insertion* operator is again applied.
5. The execution ends after a fixed number (p_b) of non-improving perturbations.

In addition, after a fixed number of iterations (p_c) the maximum window size w is increased, thus providing a deepening factor.

Algorithm 1 WINASOBS*()

```
1:  $w = 1$
2:  while  do$time()$
3:      for $i \leftarrow 1, ..., p_c$ do
4:          $order \leftarrow sampleNewOrder()$
5:          $asobs(order)$
6:          $localSearch(order, w)$
7:          for $i \leftarrow 1, ..., p_b$ do
8:              $perturb(order, p_a)$
9:              $localSearch(order, w)$
10:         end for
11:     end for
12:     $w = w + 1$
13: end while
```

IINOBS is based on the *insertion* operator coupled with a perturbation factor, and can thus be viewed as WINOBS without our novel *window insertion* operator and the intensification scheme.

5. Experiments

We consider 18 data sets with 64 to 1556 binary-valued variables, commonly used in the literature of structure learning, firstly introduced in (Lowd and Davis, 2010) and (Haaren and Davis, 2012). Originally each data set has been split into three subsets of instances. This yields 54 data sets, . The number of variables in each data set is reported in Table 5.

Additionally we create data sets with very large numbers of variables by generating 15 synthetic networks: five networks of size 2000, five networks of size 4000, five networks of size 10000 (Table 5). We used to this end the software BNgenerator.[1] Each variable has

1. `http://sites.poli.usp.br/pmr/ltd/Software/BNGenerator/`

a number of states randomly drawn from 2 to 4 and a number of parents randomly drawn from 0 to 6. From each network we sample a data set of 5000 instances.

For all the data sets, the cache of parent sets was pre-computed using the approximated exploration of parent set proposed in (Scanagatta et al., 2015). We then measure the BIC score of the best Bayesian networks found by each approach. Let us denote by $\Delta BIC_{1,2} = \Delta BIC_1 - \Delta BIC_2$ the difference between the BIC score of network 1 and network 2. Positive values of $\Delta BIC_{1,2}$ imply evidence in favor of network 1. The difference in BIC score can be mapped on the Bayes factor (Raftery, 1995, Tab. 6) and interpreted accordingly. For instance the evidence in favor of network 1 is respectively weak, positive, strong, very strong if $\Delta BIC_{1,2}$ is between 0 and 2; 2 and 6; 6 and 10 ; beyond 10.

	n		n		n
Kdd	64	Retail	135	EachMovie	500
Plants	69	Pumsb-star	163	WebKB	839
Audio	100	DNA	180	Reuters-52	889
Jester	100	Kosarek	190	C20NG	910
Netflix	100	MSWeb	294	BBC	1058
Accidents	111	Book	500	Ad	1556

Table 2: Real data sets used in the experimental validation.

	n		n		n
R2-0	2000	R4-0	4000	R10-0	10000
R2-1	2000	R4-1	4000	R10-1	10000
R2-2	2000	R4-2	4000	R10-2	10000
R2-3	2000	R4-3	4000	R10-3	10000
R2-4	2000	R4-4	4000	R10-4	10000

Table 3: Syntethic data sets used in the experimental validation

5.1 Improving ASOBS with the operators

We first assess the application of three operators as a post-processing step to the solutions proposed to ASOBS. We thus compare *swap*, *insertion* and *window* (with maximum window size of 5). We allow one hour of computation to each approach. We record the scores of all the networks generated by the algorithm during its various iterations. We report in Table 4 the summary regarding the best and the mean scores of the solutions found, and in Figure 6 the summary regarding the number of solutions $\mathcal{S}$ proposed.

The *window* operator clearly improves over the plain ASOBS. It consistently yields a higher BIC score both for the mean and the maximum; moreover, in all the data sets, the score of one random ordering improved by the *window insertion* was higher than **all** the orderings sampled by ASOBS in one hour.

The *window insertion* yields also mean score and best score consistently higher than the *swap* operator; the ΔBIC is higher than 100 in all the 54 data sets. The number of total iterations on the other hand is significantly lower, showing the trade-off between quality of the scores and required time for the exploration.

Also the comparison between the window insertion operator and the *insertion* operator is consistently in favor of window insertion, both as for the best and the mean score.

Table 4: Comparison between ASOBS with window and ASOBS with other operators: plain (no operator), swap, insertion.

	ΔBIC					
	wind. vs **plain**		**wind.** vs **swap**		**wind.** vs **ins**	
	Best	Mean	Best	Mean	Best	Mean
Very positive (K >10)	54	54	54	54	45	51
Strongly positive (6<K<10)	0	0	0	0	0	0
Positive (2<K<6)	0	0	0	0	0	0
Neutral (-2<K<2)	0	0	0	0	9	3
Negative (-6<K<-2)	0	0	0	0	0	0
Strongly negative (-10<K<-6)	0	0	0	0	0	0
Very negative (K<-10)	0	0	0	0	0	0

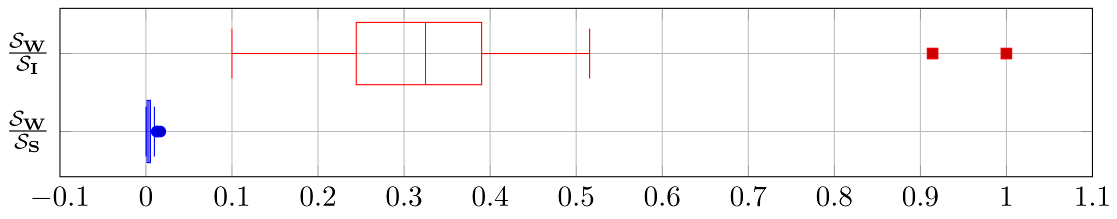

Figure 6: Ratio between the number of iterations (i.e., structures found) of ASOBS coupled with window and with insertion; with window and insertion.

We report in Fig.6 the distribution across data sets of $\frac{\mathcal{S}_\mathbf{W}}{\mathcal{S}_\mathbf{S}}$ and $\frac{\mathcal{S}_\mathbf{W}}{\mathcal{S}_\mathbf{I}}$, where $\mathcal{S}_\mathbf{W}$, $\mathcal{S}_\mathbf{S}$ and $\mathcal{S}_\mathbf{I}$ are the number of solution generated respectively by ASOBS with window, swap and insertion operators. The window insertion is a more demanding operator than its predecessors and as such it performs on average only 30% of the iterations done by insertions and only about 3% of the iterations done by the swap operator (Fig. 6).

We call WINASOBS the package resulting by coupling ASOBS with the window operator.

5.2 Exploration meta-heuristic

We compare WINASOBS against two recently proposed algorithms for Bayesian network structure learning: MINOBS and IINOBS (Lee and van Beek, 2017). They have been shown (Lee and van Beek, 2017) to outperform many other greedy hill-climbing local search over orderings, and also Gobnilp (Cussens, 2011), the state-of-the-art exact solver. We used the configuration of parameters suggested by the authors and their implementation, available online.[2]

Each method was executed three times, and we report the mean of the best solutions found in each execution. In Table 5 we summarize the results obtained on the real data sets (whose number of variable varies between 64 and 1556), and in Table 5 we summarize the results obtained on the syntethic data sets (whose number of variables ranges between 2000 and 10000). We denote by $\Delta BIC_{\mathbf{W},\mathbf{I}}$ the difference in BIC score between the network found by WINASOBS and IINOBS; we denote by $\Delta BIC_{\mathbf{W},\mathbf{M}}$ the difference in BIC score between the network found by WINASOBS and MINOBS.

In (Lee and van Beek, 2017), the authors indicate that IINOBS is the better alternative when time is limited, and thus is the competitor for short execution times. They further state that MINOBS has a slower improvement curve, but eventually outperforms IINOBS, thus is the competitor for long execution times. To verify the sensitivity of the results on the allowed time limit, we report the ΔBIC of the best solution found after 1 hour, 6 hours and 12 hours.

Table 5: Comparison between WINASOBS , IINOBS and MINOBS and on the real data sets.

	$\Delta BIC_{\mathbf{W},\mathbf{I}}$			$\Delta BIC_{\mathbf{W},\mathbf{M}}$		
	1 hour	6 hours	12 hours	1 hour	6 hours	12 hours
Very positive (K >10)	27	20	21	33	16	15
Strongly positive (6<K<10)	4	7	0	2	0	0
Positive (2<K<6)	8	6	8	7	2	5
Neutral (-2<K<2)	9	11	16	6	18	16
Negative (-6<K<-2)	3	7	4	4	3	4
Strongly negative (-10<K<-6)	0	0	0	0	0	2
Very negative (K<-10)	3	3	5	2	15	12

Let us start from the real data sets, whose number of variables ranges up to 1556. WINASOBS outperforms IINASOBS in all the time limits. It should be noted that the difference between them is a) the adoption of ASOBS instead of OBS; b) the application of the window operator instead of the insertion operator. The comparison with MINOBS shows that if the computational time is limited to 1 hour, WINASOBS performs better than MINOBS; as the computational time increases to 6 and 12 hours, there is no clear winner between the two methods.

The comparison on the synthetic data sets (whose number of variables ranges between 2000 and 10000) shows WINASOBS as the consistent winner compared to *both* INOBS and MINOBS, for *all* the considered time limits. With the largest number of variables, MINOBS often failed to deliver even one starting solution when the time limit was less than 6 hours. This confirms the observation made in (Lee and van Beek, 2017) that MINOBS struggles with the largest dataset.

2. `https://cs.uwaterloo.ca/~vanbeek/Research/research_ml.html`

Table 6: Comparison between WINASOBS , IINOBS and MINOBS and on the synthetically generated data sets.

	$\Delta BIC_{\mathbf{W},\mathbf{I}}$			$\Delta BIC_{\mathbf{W},\mathbf{M}}$		
	1 hour	6 hours	12 hours	1 hour	6 hours	12 hours
Very positive (K >10)	15	15	14	15	14	13
Strongly positive (6<K<10)	0	0	0	0	0	0
Positive (2<K<6)	0	0	0	0	0	0
Neutral (-2<K<2)	0	0	1	0	1	1
Negative (-6<K<-2)	0	0	0	0	0	1
Strongly negative (-10<K<-6)	0	0	0	0	0	0
Very negative (K<-10)	0	0	0	0	0	0

6. Conclusions

We presented an operator for greedy hill-climbing in the space of the orderings. We couple it with an ordering-based algorithm (ASOBS) for structural learning of Bayesian networks. We call WINASOBS the resulting approach. We compare it to the state-of-the-art approaches on a wide range of problems; experimentally it outperforms the competitors when the allowed execution times is limited to up to 6 hours, or when the number of variables is greater than one thousand.

Acknowledgments

Work partially supported by the Swiss NSF grants Nos. 200021_146606 / 2, IZKSZ2_162188.

References

Juan I. Alonso-Barba, L. de la Ossa, and Jose M. Puerta. Structural learning of Bayesian networks using local algorithms based on the space of orderings. *Soft Computing*, 15(10): 1881–1895, 2011.

M. Ben-Daya and M. Al-Fawzan. A tabu search approach for the flow shop scheduling problem. *European journal of operational research*, 109(1):88–95, 1998.

D. M. Chickering, C. Meek, and D. Heckerman. Large-sample learning of Bayesian networks is hard. In *Proceedings of the 19st Conference on Uncertainty in Artificial Intelligence*, UAI-03, pages 124–133. Morgan Kaufmann, 2003.

J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27st Conference Annual Conference on Uncertainty in Artificial Intelligence*, UAI-11, pages 153–160. AUAI Press, 2011.

C. P. de Campos and Q. Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

C. P. de Campos, Z. Zeng, and Q. Ji. Structure learning of Bayesian networks using constraints. In *Proceedings of the 26st Annual International Conference on Machine Learning*, ICML-09, pages 113–120, 2009.

J. V. Haaren and J. Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the 26st AAAI Conference on Artificial Intelligence*, 2012.

T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian Network Structure using LP Relaxations. In *Proceedings of the 13st International Conference on Artificial Intelligence and Statistics*, AISTATS-10, pages 358–365, 2010.

M. Koivisto and K. Sood. Exact Bayesian Structure Discovery in Bayesian Networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

C. Lee and P. van Beek. Metaheuristics for score-and-search Bayesian network structure learning. In *Proceedings of the 30th Canadian Conference on Artificial Intelligence*, 2017.

D. Lowd and J. Davis. Learning Markov network structure with decision trees. In *Proceedings of the 10st Int. Conference on Data Mining*, ICDM2010, pages 334–343, 2010.

A. E. Raftery. Bayesian model selection in social research. *Sociological methodology*, 25: 111–164, 1995.

M. Scanagatta, C. P. de Campos, G. Corani, and M. Zaffalon. Learning Bayesian Networks with Thousands of Variables. In *Advances in Neural Information Processing Systems 28*, NIPS-15, pages 1855–1863, 2015.

T. Silander and P. Myllymaki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, UAI-06, pages 445–452, 2006.

M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, UAI-05, pages 584–590, 2005.

P. van Beek and HF. Hoffmann. Machine learning of Bayesian networks using constraint programming. In *21st International Conference on Principles and Practice of Constraint Programming*, CP-15, pages 429–445, 2015.

C. Yuan and B. Malone. An improved admissible heuristic for learning optimal Bayesian networks. In *Proceedings of the 28st Conference on Uncertainty in Artificial Intelligence*, UAI-12, 2012.

C. Yuan and B. Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.