Metaheuristics for Score-and-Search Bayesian Network Structure Learning

Colin Lee and Peter van Beek

Cheriton School of Computer Science, University of Waterloo

Abstract. Structure optimization is one of the two key components of score-and-search based Bayesian network learning. Extending previous work on ordering-based search (OBS), we present new local search methods for structure optimization which scale to upwards of a thousand variables. We analyze different aspects of local search with respect to OBS that guided us in the construction of our methods. Our improvements include an efficient traversal method for a larger neighbourhood and the usage of more complex metaheuristics (iterated local search and memetic algorithm). We compared our methods against others using test instances generated from real data, and they consistently outperformed the state of the art by a significant margin.

1 Introduction

A Bayesian network is a probabilistic graphical model which encodes a set of random variables and their probabilistic relationships through a directed acyclic graph. Bayesian networks have been successfully applied to perform tasks such as classification, knowledge discovery, and prediction in fields including medicine, engineering, and business [15]. Bayesian network structure learning involves finding the acyclic graph that best fits a discrete data set over the random variables.

Bayesian networks can be learned through the method of score-and-search. In score-and-search a scoring function indicates how well a network fits the discrete data and search is used to find a network which achieves the best possible score by choosing a set of parents for each variable. Unfortunately, finding such a network is $\mathcal{NP}$-hard, even if each node in the network has at most two parents [11]. Exact solvers for this problem have been developed using a variety of techniques (e.g., [2,3,7,20]). These methods achieve good performance on smaller instances of the problem but fail to scale in terms of memory usage and runtime on instances with more than 100 variables unless the indegree is severely restricted.

Local search has been shown to be successful in finding high quality solutions to hard combinatorial problems with relatively simple algorithms [9]. It has already been applied to learning Bayesian networks using techniques such as greedy search [5], tabu search [19], and ant colony optimization [8], over search spaces such as the space of network structures [5], the space of equivalent network structures [4], and the space of variable orderings [12,19]. We improve upon the approach of ordering-based search (OBS) by Teyssier and Koller [19], which

makes use of the topological orderings of variables as a search space. Teyssier and Koller [19] show that OBS performs significantly better than local search over network structures on this problem.

In this paper, we make the following contributions. We identify the Bayesian network structure learning problem as being similar to the Linear Ordering Problem (LOP) and adapt the local search techniques applied to LOP in [17] to improve OBS. First, as previously done in [1], we experiment with using a neighbourhood which is larger than the one originally used in OBS to find high quality local optima. We then include optimizations to make the use of this neighborhood more feasible for instances with a high number of variables. We combine our local search method with iterated local search (ILS) and memetic algorithm (MA) to produce two new methods. Experimental results show that the new methods are able to find networks that score significantly better than other state of the art anytime solvers on instances with hundreds of variables.

2 Background

A Bayesian network is composed of a directed acyclic graph (DAG) G with random variables $\{X_1, \ldots, X_n\}$ as vertices. The score-and-search method of Bayesian network learning makes use of a scoring function $sc(G \mid I)$ which takes a set $I = \{I_1, \ldots, I_n\}$ of complete instantiations I_i of the variables (the data) and assigns a real valued score to the network G. For the purposes of this paper, a lower score will indicate a higher quality network. Also, the data parameter I will be made implicit, so that we write $sc(G)$ instead of $sc(G \mid I)$.

The score-and-search method consists of two stages. The first stage, called *parent set identification*, consists of computing the scores of sets of parents for each variable. A scoring function is *decomposable* if the score $sc(G)$ of the network can be written as the sum of its local scores $\sum_{i=1}^{n} sc_i(Pa(X_i))$, where $Pa(X_i)$ is the set of parents of X_i in G. Commonly used scoring functions, including BIC and BDeu which we use in our experiments, are decomposable [2]. In practice, the indegree for each variable is often bounded by some small integer k to increase the speed of inference [11]. As is usual in score-and-search approaches, we assume precomputed caches are available using techniques for efficiently scoring and pruning parent sets [16,19], resulting in a *cache* C_i of c_i candidate parent sets for each variable X_i along with their associated scores. More formally, for each i, we have $C_i \subseteq \{U : U \subseteq \{X_1, \ldots, X_n\} \setminus \{X_i\}, |U| \leq k\}$, from which $sc_i(U)$ for $U \in C_i$ can be queried in constant time.

Our work focuses on the second component of score-and-search, called *structure optimization*. This involves searching for a network structure which achieves the minimum possible score by selecting a parent set for each variable X_i from C_i such that the graph is acyclic. Following previous work, we apply local search over the space of orderings of variables [19], as the space of variable orderings is significantly smaller than the space of all possible DAGs. We call a Bayesian network G *consistent* with an ordering O of its variables if O is a topological ordering of G. For a given ordering, using a bitset representation we can find the

Algorithm 1: Hill climbing for ordering-based structure optimization.

Result: A local minimum in the neighbourhood defined by *neighbours*

$O \leftarrow randomOrdering()$;
$curScore \leftarrow sc(O)$;
while *neighbours(O) contains an ordering which improves curScore* **do**
    $O \leftarrow selectImprovingNeighbour(O)$;
    $curScore \leftarrow sc(O)$;
return O

optimal parent sets for all of the variables in $O(Cn)$ operations, where C is the total number of candidate parent sets. Thus, the problem of finding the optimal network can be transformed into finding the ordering with the lowest score.

3 Search Neighbourhood

We start by building upon the hill climbing method that is used in OBS (see Alg. 1). This method consists of first randomly generating an ordering O and computing its score. Then, until no ordering in the *neighbourhood* of O has a higher score than O, O is set to one of its neighbours with an improving score. O will then be a local minimum in the neighbourhood. We call each iteration to an improving neighbour a hill climbing *step*.

The choice of neighbourhood significantly impacts the performance of hill climbing, as it essentially defines the search landscape. In OBS, the *swap-adjacent* neighbourhood is used. Formally, for an ordering $O = (X_1, \ldots, X_n)$, O' is a neighbour of O if and only if $O' = (X_1, \ldots, X_{i+1}, X_i, \ldots, X_n)$ for some $1 \leq i \leq n-1$. The size of the neighbourhood is therefore $n-1$. In hill climbing, other than in the first step, the optimal parents sets of O are already computed from the previous step. Subsequently, the optimal parent sets only need to be updated for X_i and X_{i+1} as the swap in the ordering does not affect the potential parents of other variables. Therefore, the cost of checking the score of a neighbour defined by the swap-adjacent neighbourhood is $O((c_i + c_{i+1})n)$. The total cost of computing the score of all neighbours of O (a *traversal* of the neighbourhood) is then $O(Cn)$. From [16], a further optimization can be made by checking for an updated parent set for X_{i+1} only if X_i was one of its parents before the swap. Additionally, when updating X_i's parent set after the swap, only the candidate parent sets that contain X_{i+1} needs to be considered for an improvement.

A pitfall of the swap-adjacent neighbourhood is a high density of weak local minima. Swapping adjacent variables X_i and X_{i+1} does not have a large impact on their parent sets, as X_{i+1} only loses the ability to have X_i as a parent and X_i only gains the ability to have X_{i+1} as a parent. Given that the parent set size is restricted to k, which is significantly smaller than n in practice, it is unlikely that an adjacent swap will lead to an improvement in the score. In the terminology of local search, using this neighbourhood results in a search landscape with large

$$
\begin{aligned}
O = &(X_1, X_2, X_3, X_4, X_5, X_6) && \rightarrow_{swap} \\
&(X_1, X_3, X_2, X_4, X_5, X_6) && = (\text{Insert } X_2 \text{ in } O \text{ to index } 3) \rightarrow_{swap} \\
&(X_1, X_3, X_4, X_2, X_5, X_6) && = (\text{Insert } X_2 \text{ in } O \text{ to index } 4) \rightarrow_{swap} \\
&(X_1, X_3, X_4, X_5, X_2, X_6) && = (\text{Insert } X_2 \text{ in } O \text{ to index } 5) \rightarrow_{swap} \\
&(X_1, X_3, X_4, X_5, X_6, X_2) && = (\text{Insert } X_2 \text{ in } O \text{ to index } 6) \rightarrow_{swap}
\end{aligned}
$$

Fig. 1: Example of performing the four forward inserts of X_2 for O using four swap-adjacent moves. The final insert left for X_2 (to index 1) can be achieved with a similar swap-adjacent move in O with X_1. Adapted from [17].

plateaus. OBS attempts to alleviate this problem by using a tabu list which allows the traversal of the search over non-improving solutions.

Keeping this in mind, we consider the *insert* neighbourhood, which contains orderings that can be obtained from one another by selecting a variable and inserting it at another index. Formally, $O = (X_1, \ldots, X_{i-1}, X_i, X_{i+1}, \ldots, X_{j-1}, X_j, X_{j+1}, \ldots, X_n)$ is neighbouring O' in the insert neighbourhood if and only if $O' = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_{j-1}, X_j, X_i, X_{j+1}, \ldots, X_n)$, for some i and j. We say O' is O with the variable X_i inserted into position j. The use of the insertion neighbourhood as an improvement to OBS is explored in [1], but there it is used with limited success.

After inserting variable X_i to index j, the possible parents for all variables from X_i to X_j (inclusive) in the original ordering have been updated, and the optimal parent set for each one of these must be checked. In the case that for $i < j$, this takes $O((\sum_{i=1}^{j} c_i)n)$ operations. The case for $j < i$ is the same but with the indices swapped. Naïvely computing the scores of all $(n-1)^2$ neighbours independently therefore has cost $O(Cn^3)$, which is significantly greater than the $O(Cn)$ cost required for traversing the swap-adjacent neighbourhood.

Fortunately, as shown in [6] and applied to OBS in [1], the insert neighbourhood can be traversed with a series of $O(n^2)$ swap-adjacent moves. Specifically, given a variable X_i, the $n-1$ neighbours of O formed by inserting X_i into one of the $n-1$ other indices can be constructed with a series of $n-1$ swap-adjacent moves (see Fig. 1). Since a score update for a swap-adjacent move can be done in $O((c_i + c_{i+1})n)$ operations, the cost to compute the scores of the ordering for all $n-1$ indices that X_i can be inserted into is $O(Cn)$. There are n choices for X_i, so the cost of traversing the entire neighbourhood is $O(Cn^2)$. Along with being an order of magnitude faster than the naïve traversal, the previously mentioned optimizations for scoring swap-adjacent moves can be applied in this method.

Even in small cases, local minima in the swap-adjacent neighbourhood can be overcome by using an insert move (see Fig. 2). The swap-adjacent neighbourhood of an ordering is a subset of the insert neighbourhood of that ordering. Thus, the lowest scoring neighbour in the insert neighbourhood is guaranteed to score at least as low as any neighbour from the swap-adjacent neighbourhood.

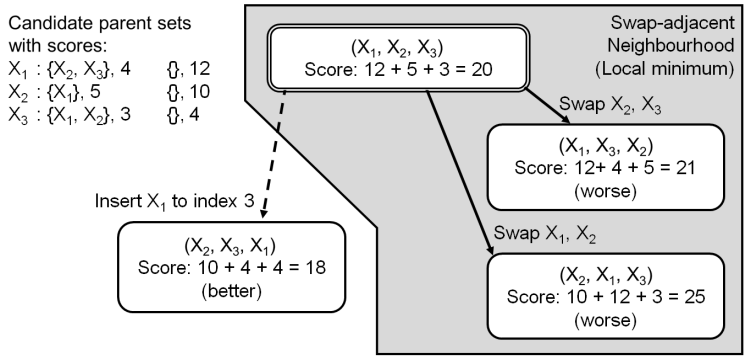

Fig. 2: Example local minimum for the swap-adjacent neighbourhood. A local minimum occurs at the ordering $O = (X_1, X_2, X_3)$. The two swap-adjacent neighbours of O are indicated with solid arrows; both have worsening scores. An insert move neighbour, indicated with a dashed arrow, gives an improved score. Hence O is not a local minimum in the insert neighbourhood.

Three different possibilities for neighbour selection were tested for the insert neighbourhood: best improvement, first improvement, and hybrid.

- *Best improvement*: The neighbour with the highest score is chosen. Finding the score of every neighbour takes $O(Cn^2)$ operations.
- *First improvement*: The insert moves are evaluated in random order and the first neighbour with an improving score is selected. Finding the score of each random neighbour costs $O(Cn)$ operations each, so that in the worst case, where every neighbour must be scored, $O(Cn^3)$ operations are used.
- *Hybrid*: This selection scheme falls between best and first improvement and is adapted from [17]. A variable X is randomly chosen from the ordering. The index j to insert X which gives the highest score is then found using $n-1$ adjacent swaps to score each index. If this insertion gives an improvement in score, then it is chosen. If no insertions for X give an improvement, another variable is randomly chosen and the process repeats. In the worst case of a local minimum, all n variables are tested for improving insert moves, using a total of $O(Cn^2)$ operations.

The three selection methods are experimentally compared in Section 6. Our results show hybrid selection to be the most appropriate. We call the hill climbing method obtained from Algorithm 1 by using the insert neighbourhood and hybrid neighbour selection method Insert Neighbourhood OBS (INOBS). INOBS is a key component of our metaheuristic methods in the following sections.

4 Iterated Local Search

Iterated local search (ILS) has historically been a simple and intuitive extension of basic hill climbing which performs competitively with other metaheuristic

Algorithm 2: An outline of an ILS algorithm.

```
1 O ← initialState();
2 O ← localSearch(O);
3 while terminationCondition(O, history) is not met do
4 |   O′ ← perturb(O);
5 |   O′ ← localSearch(O′);
6 |   O ← acceptenceCriterion(O, O′);
```

methods [9]. The improvement ILS brings over hill climbing with random restarts is the ability to continue searching for improvements nearby a good solution (local minima) instead of erasing progress by simply restarting. First, a random ordering is chosen as an initial candidate solution and a subsidiary local minimum is found through local search. Then, three basic steps are iterated over until a restart condition is met: first, the current solution is perturbed through some *perturbation operator*. Then, local search is applied to the perturbed solution to obtain a new local minimum. Whether or not the new local minimum will replace the old one before the next iteration is decided upon according to an *improvement criterion*. The iterations stop when a specified *termination condition* is met. This generic ILS algorithm is outlined in Algorithm 2.

Using INOBS as the local search component for the ILS procedure, we construct Iterated INOBS (IINOBS). For the perturbation operator of IINOBS, $p_s \cdot n$ pairs of variables were swapped by their index in the ordering, where p_s is called the perturbation factor. Swaps are chosen because they are not easily undone by insertions, so it is unlikely that the proceeding local search will reverse the perturbation [17]. As for the improvement criterion, the solution is accepted when the new local minima achieves a score s' such that $s'(1-\varepsilon) < s$, where s is the score of the original local minima and $\varepsilon \geq 0$. The parameter ε allows some leeway for the new local minima to have a worse score than the current one. To avoid stagnation, IINOBS is restarted from a new initial ordering according to both a soft and hard restart schedule. A soft restart occurs if the objective value has not been improved in over r_s moves to new local optima. A hard restart occurs when r_h moves to new local optima have occurred, regardless of how the search has been improving.

5 Memetic Algorithm

Memetic INOBS (MINOBS) is a memetic search method for the problem which uses INOBS as a local search procedure. Memetic search allows a local search algorithm to be combined with the intensification and diversification traits of population based search techniques. The method can be compared to a standard genetic algorithm except using the space of local minima rather than the space of all possible orderings [13]. An outline of the memetic algorithm we fit MINOBS into is in Algorithm 3.

Algorithm 3: An outline of the memetic algorithm that INOBS is fit into to construct MINOBS. Adopted from [17].

```
1  population ← {};
2  for i ← 1, ..., N (N is the population size) do
3  |  O ← localSearch(randomInitialState());
4  |  population ← population ∪ {O};
5  while termination condition is not met do
6  |  offspring ← {};
7  |  for i ← 1, ..., c_n do
8  |  |  choose O_1, O_2 from population;
9  |  |  offspring ← offspring ∪ localSearch(crossover(O_1, O_2))};
10 |  for i ← 1, ..., m_n do
11 |  |  choose O_1, O_2 from population;
12 |  |  offspring ← offspring ∪ localSearch(mutate(O_1, O_2, m_p))};
13 |  population ← prune(population ∪ offspring, N);
14 |  if the average score in the population does not change by d_Δ in the last d_t
   |   generations then
15 |  |  population ← selectBest(population, d_n);
16 |  |  for i ← 1, ..., N − d_n do
17 |  |  |  O ← localSearch(randomInitialState());
18 |  |  |  population ← population ∪ {O};
```

Memetic search roughly resembles maintaining multiple runs of ILS in parallel. Performing crossover and mutation is analogous to perturbation. Pruning the population is analogous to automatically stopping the less promising of the parallel runs. Therefore, we expect MINOBS to perform at least as well as IINOBS given a sufficient amount of time.

The algorithm begins with an initial population of random local optima are generated through INOBS. Until the termination condition is met, the population undergoes a number of generations. Each generation consists of a crossover and a mutation stage. During the crossover stage, members of the population are randomly drawn in pairs and combined according to some crossover operator to produce a new ordering. INOBS is then applied to the new orderings. In the mutation stage, random members of the population are chosen from the population and perturbed according to the swap-based perturbation operation presented in the IINOBS. The new permutations are then subjected to local search using INOBS. Afterwards, both the orderings produced in the crossover and mutations stages are added to the population. Finally, members of the population are pruned to maintain the original size of the population according to some pruning scheme. In our case, pruning involved filtering out orderings with duplicate scores and then afterwards removing orderings with the lowest scores until the population was back to its original size.

MINOBS also has the possibility of performing a diversification step if the average score of the population does not change by over d_Δ for d_t generations. The diversification step consists of removing all but the top d_n members of the population and refilling the rest of the population with new random local minima. The diversification step's purpose is to stop the population from stagnating and acts similarly to a random restart. We experimented with three different crossover operators. Let $O_1 = (X_{\pi_1(1)}, \ldots, X_{\pi_1(n)})$ and $O_2 = (X_{\pi_2(1)}, \ldots, X_{\pi_2(n)})$ be the two orderings to cross to produce the offspring $O = (X_{\pi(1)}, \ldots, X_{\pi(n)})$, where π_1, π_2, and π are permutations of indices from 1 to n.

- *Cycle crossover* (CX): A random index i is selected along with a random parent O_1, without loss of generality. For the resulting ordering O, we set $\pi(i)$ as $\pi_1(i)$. Then for the other parent O_2, the index j such that $\pi_1(j) = \pi_2(i)$ is found, and we set $\pi(i)$ to $\pi_1(j)$. Index i is then set to j and the process repeats until i cycles back to the original index. The process then restarts with i as an index unused by π until π is completed. The resulting permutation π has the property that $\pi(i) = \pi_1(i)$ or $\pi(i) = \pi_2(i)$ for every index i [14].
- *Rank crossover* (RX): The offspring is based on sorting the mean index of each variable over both parent orderings. When a ties occur (two or more elements share the same mean index over both parents), the order of the elements is determined according to a random distribution [17].
- *Order-based crossover* (OB): From O_1, $n/2$ variables are randomly chosen and copied in the same position into offspring O. The variables not copied from O_1 fill the unused positions in O according to their order in O_2 [18].

6 Experimental Results

We report on experiments to (i) evaluate the three neighbourhood selection schemes, (ii) select the parameters for our two proposed metaheuristic methods, IINOBS and MINOBS, and (iii) compare our INOBS, IINOBS and MINOBS methods against the state of the art for Bayesian network structure learning.

Most of the instances used in our experiments were provided by the Bayesian Network Learning and Inference Package (BLIP)[1]. These instances used the BIC scoring method and have a maximum indegree of $k = 6$. Other instances were produced from datasets from J. Cussens and B. Malone and scored using code provided by B. Malone. The method of scoring for an instance in this set (BIC or BDeu) is indicated in the instance name.

Experiment 1. In the first set of experiments, we compared the three neighbourhood selection schemes—best improvement, first improvement, and hybrid—incorporated into the basic hill climbing algorithm (Alg. 1), which terminates once a local minima is reached (see Table 1). Overall, best and first improvement generated higher quality local minima than hybrid selection. Note that it is possible for first improvement to be better than best improvement as they follow a different trajectory through the search space from the first move. Best

[1] `http://blip.idsia.ch/`

Table 1: Average runtime (sec.) and score for each neighbourhood selection method for various benchmarks (100 runs), where n is the number of variables in the instance and C is the total number of candidate parent sets.

			Best		First		Hybrid	
instance	n	C	time	score	time	score	time	score
segment_BIC	20	1053	0.01	15176.3	0.01	15175.9	0.00	15176.8
autos_BIC	26	2391	0.02	1585.5	0.03	1586.4	0.01	1587.1
soybean_BIC	36	5926	0.08	3155.9	0.08	3156.5	0.02	3158.4
wdbc_BIC	31	14613	0.19	6623.0	0.16	6624.6	0.04	6627.0
steel_BDeu	28	113118	2.09	18690.0	1.76	18674.8	0.59	18685.9
baudio.ts	100	371117	23.76	194711.9	12.63	193795.5	3.03	194922.0
jester.ts	100	531961	36.56	88098.9	19.86	87271.6	4.48	87871.1
tretail.ts	134	435976	36.54	106864.0	16.59	106294.6	3.03	106879.2
munin-5000	1041	1648338	847.08	1041284.5	961.39	1041219.6	17.97	1040638.6

improvement for insertion-based OBS is the best method tested in [1] where it is called HCbO. The authors note that the performance of their algorithm is hindered by the need to evaluate the scores of all $(n-1)^2$ neighbours, even with the optimizations mentioned in Section 3. They attempted to lower the neighbourhood size by restricting the insert radius and using variable neighbourhood search, but the modifications were not effective in improving performance.

While overall hybrid selection performed marginally worse in terms of score, it took significantly less time to reach a local minima on the larger instances, scaling to about fifty times faster on the largest instance. One further optimization for best improvement with insert moves is explored in [1] but was not implemented in our experiments. However, this optimization only improves the speed by at most a factor of two, which is still not enough to make best improvement comparable to hybrid. Following the note that best improvement selection is too time consuming in [1] and our own focus on scaling to larger instances, we designated hybrid as our neighbourhood selection method of choice for our metaheuristic methods to make a direct improvement over HCbO.

Experiment 2. In the second set of experiments, we tuned the parameters for our two proposed metaheuristic methods, IINOBS and MINOBS. Parameter tuning was performed with ParamILS, a local search based tuning method for metaheuristics [10]. Tuning was performed using three instances from the BLIP benchmarks (*baudio.ts*, *jester.ts*, *tretail.ts*). Unfortunately, larger instances could not be used effectively for tuning due to time constraints. The objective minimized by ParamILS was the mean percent difference from the best scores generated by INOBS with random restarts. The parameters tuned and their optimal value found by ParamILS are listed in Table 2 and Table 3. (Near-optimal parameters were also experimented with and similar results were obtained.)

Experiment 3. In the final set of experiments, we compared our two proposed metaheuristic methods, IINOBS and MINOBS, as well as INOBS with random restarts, to the state of the art (see Table 4). For the state of the art, we compare

Table 2: Parameters tuned for IINOBS.

parameter	description	value
p_s	Perturbation factor: $p_s \cdot n$ random swaps will be used to perturb the ordering.	0.03
ε	Leeway allowed when choosing a local minima to move to.	0.00005
r_s	Number of non-improving steps until a restart.	22
r_h	Number of perturbations until a restart.	100

Table 3: Parameters tuned for MINOBS.

parameter	description	value
N	Number of members in the population.	20
crossover	Type of crossover to perform.	OB
c_n	Number of crossovers to perform.	20
m_p	Mutation power factor: $m_p \cdot n$ random swaps will be used to perturb the ordering.	0.01
m_n	The number of mutations to perform.	6
d_t	Number of scores to look back for triggering a diversification step.	32
d_Δ	Max. change in ave. score needed to trigger a diversification step.	0.001
d_n	Number of members to keep after a diversification step.	4

against (i) our implementation of OBS [19]; (ii) GOBNILP (v1.6.2) [7], an exact solver used here as an anytime solver that was run with its default parameters; and (iii) acyclic selection OBS (ASOBS [16], a recently proposed local search solver that has been shown to out-perform all competitors on larger instances. OBS, INOBS, INOBS with restarts, IINOBS, and MINOBS were implemented in C++ using the same code and optimizations for swap-adjacent moves[2]. ASOBS is written in Java and is therefore expected to run more slowly compared to our methods. However, our experiment runtime is long enough for ASOBS to stagnate enough so that even if the implementation of the method was several times faster, it is unlikely that the results would change significantly.

Experiments for all methods other than ASOBS were run on a single core of an AMD Opteron 275 @ 2.2 GHz. Each test was allotted a maximum 30 GB of memory and run for 12 hours. The generation of the instances from data can take days, so this time limit is reasonable. Due to limited software availability, tests for ASOBS were run courtesy of M. Scanagatta with the same time and memory limits and on a single core of an AMD Operton 2350 @ 2 GHz. These two processors have similar single core performance. We used test instances from the BLIP benchmarks. Of the 20 data sets used to generate the BLIP instances, three were excluded because they were used in tuning, leaving 17 for testing. Three additional large instances were chosen that were generated with data from real networks (*diabetes-5000*, *link-5000*, *pigs-5000*). Excluding ASOBS, three tests with different random seeds were tested for each instance-method pair, and the median was recorded. ASOBS was only run once due to time constraints.

[2] The software is available at: `https://github.com/kkourin/mobs`

Table 4: Median score of best networks found, for various benchmarks, where n is the number of variables and C is the total number of candidate parent sets. The column labelled INOBS represents the scores of INOBS with random restarts. OM indicates the solver runs out of memory before any solution is output. Bold indicates the score was the best found amongst all tested methods. An asterisk indicates that the score is known to be optimal.

instance	n	C	GOBNILP	OBS	ASOBS	INOBS	IINOBS	MINOBS
nltcs.test	16	48303	**5836.6***	5903.4	**5836.6***	**5836.6***	**5836.6***	**5836.6***
msnbc.test	17	16594	**151624.7***	153291.6	**151624.7***	**151624.7***	**151624.7***	**151624.7***
kdd.test	64	152873	57271.3	57556.2	57522.6	57218.0	**57209.6***	**57209.6***
plants.test	69	520148	19337.8	16485.0	16681.4	14649.6	**14539.7**	**14539.7**
bnetflix.test	100	1103968	OM	13033.3	12545.1	12282.4	**12279.8**	**12279.8**
accidents.test	111	1425966	OM	3454.4	2119.9	855.9	**828.3**	**828.3**
pumsb_star.test	163	1034955	11552.5	5626.9	3641.7	3068.5	**3062.8**	**3062.8**
dna.test	180	2019003	OM	21783.0	19335.1	18455.1	18297.0	**18287.8**
kosarek.test	190	1192386	OM	29283.4	26718.5	24816.5	**24731.8**	24745.9
msweb.test	294	1597487	OM	28496.3	26061.6	25781.7	25743.5	**25741.6**
book.test	500	2794588	OM	36133.0	33104.4	30614.2	30355.2	**30345.0**
tmovie.test	500	2778556	OM	8547.6	6312.4	5008.5	4765.5	**4763.8**
cwebkb.test	839	3409747	OM	34837.9	21948.7	17984.7	17564.7	**17556.4**
cr52.test	889	3357042	OM	28187.2	16060.2	13374.3	13063.0	**13013.9**
c20ng.test	910	3046445	OM	109950.7	79093.8	69832.9	69139.5	**69024.0**
bbc.test	1058	3915071	OM	44663.6	30261.3	25263.5	24498.2	**24403.9**
ad.test	1556	6791926	OM	10845.0	8745.2	7814.5	**7610.4**	7646.0
diabetes-5000	413	754563	OM	2043150.9	1925441.6	1913319.6	**1912286.3**	1912670.9
pigs-5000	441	1984359	OM	1010120.7	905538.2	802293.5	782105.9	**775953.3**
link-5000	724	3203086	OM	85516.2	43072.1	37067.2	36758.9	**36715.3**

GOBNILP, OBS, and ASOBS performed significantly worse than our proposed metaheuristic methods, IINOBS and MINOBS. A closer look at the experimental data revealed that the best solutions found by OBS and ASOBS on all but three of the smaller instances over an entire 12 hour run scored worse than the solutions found by INOBS with random restarts in a *single* hill climb. MINOBS found equivalent or better structures than IINOBS for 17/20 instances, though seven were tied. The time and score data showed that MINOBS tended to start slow but overtime managed to outperform IINOBS on most instances. This behaviour is expected if memetic search is seen as running ILS in parallel, as speculated earlier. One of the cases where MINOBS found worse solutions than IINOBS was *ad.test*, one of the biggest instances we tested with. On this instance, neither method seemed close to stagnating at the 12 hour timeout, so we reran the experiment with a time limit of 72 hours. MINOBS eventually overtook IINOBS after about 24 hours In general, IINOBS seems to be the better method if time is limited, but it begins stagnating earlier than MINOBS.

7 Conclusions

We present INOBS, IINOBS, and MINOBS: three new ordering-based local search methods for Bayesian network structure optimization which scale to hun-

dreds of variables and have no restrictions on indegree. We compare these methods to the state of the art on a wide range of instances generated from real datasets. The results indicated that these new methods are able to outperform the few score-and-search learning methods that can operate on instances with hundreds of variables. MINOBS appeared to find the best scoring network structures, with IINOBS closely following.

References

1. Alonso-Barba, J.I., de la Ossa, L., Puerta, J.M.: Structural learning of Bayesian networks using local algorithms based on the space of orderings. Soft Computing 15, 1881–1895 (2011)
2. van Beek, P., Hoffmann, H.F.: Machine learning of Bayesian networks using constraint programming. In: Proc. of CP. pp. 428–444 (2015)
3. de Campos, C.P., Ji, Q.: Efficient structure learning of Bayesian networks using constraints. J. of Machine Learning Research 12, 663–689 (2011)
4. Chickering, D.M.: Learning equivalence classes of Bayesian network structures. J. of Machine Learning Research 2, 445–498 (2002)
5. Chickering, D.M., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. In: Proc. of UAI. pp. 80–89 (1997)
6. Congram, R.K.: Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation. Phd thesis, U. of Southampton (2000)
7. Cussens, J.: Bayesian network learning with cutting planes. In: Proc. of UAI. pp. 153–160 (2011)
8. De Campos, L.M., Fernandez-Luna, J.M., Gámez, J.A., Puerta, J.M.: Ant colony optimization for learning Bayesian networks. J Approx Reason 31, 291–311 (2002)
9. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier (2004)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. JAIR 36, 267–306 (2009)
11. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. The MIT Press (2009)
12. Larranaga, P., Kuijpers, C., Murga, R., Yurramendi, Y.: Learning Bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Transactions on System, Man and Cybernetics 26, 487–493 (1996)
13. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. rep., Caltech (1989)
14. Oliver, I., Smith, D., Holland, J.R.: Study of permutation crossover operators on the TSP. In: Proc. of Int'l Conf. on Genetic Algorithms (1987)
15. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
16. Scanagatta, M., de Campos, C.P., Corani, G., Zaffalon, M.: Learning Bayesian networks with thousands of variables. In: Proc. of NIPS. pp. 1864–1872 (2015)
17. Schiavinotto, T., Stützle, T.: The linear ordering problem: Instances, search space analysis and algorithms. J. of Math. Model. and Algorithms 3, pp. 367–402 (2004)
18. Syswerda, G.: Schedule optimization using genetic algorithms. Handbook of Genetic Algorithms, pp. 332–349 (1991)
19. Teyssier, M., Koller, D.: Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In: Proc. of UAI. pp. 548–549 (2005)
20. Yuan, C., Malone, B., Wu, X.: Learning optimal Bayesian networks using A* search. In: Proc. of IJCAI. pp. 2186–2191 (2011)