Scalable Bayesian Network Structure Learning via Maximum Acyclic Subgraph

Pierre Gillot PIERRE.GILLOT@UIB.NO
Department of Informatics, University of Bergen, Norway

Pekka Parviainen PEKKA.PARVIAINEN@UIB.NO
Department of Informatics, University of Bergen, Norway

Abstract

Learning the structure of a Bayesian network is an NP-hard problem and exact learning algorithms that are guaranteed to find an optimal structure are not feasible with large number of variables. Thus, large-scale learning is usually done using heuristics that do not provide any quality guarantees. We present a heuristic method that scales up to networks with hundreds of variables and provides quality guarantees in terms of an upper bound for the score of the optimal network. The proposed method consists of two parts. First, we simplify the problem by approximating local scores using so-called edge scores. With the edge scores learning an optimal Bayesian network structure is equivalent to finding the maximum acyclic subgraph. Second, we solve the maximum acyclic subgraph problem fast using integer linear programming. Additionally, we choose the approximation in a specific way so that an upper bound for the score of an optimal network can be obtained.

Keywords: Bayesian networks; Structure learning; Integer Linear Programming.

1. Introduction

Bayesian networks are a type of probabilistic graphical model widely used in machine learning. They were originally introduced by Pearl (1988). Bayesian networks consist of two parts: a structure and parameters. The structure, represented by a directed acyclic graph (DAG), expresses conditional independencies between variables and the parameters specify local conditional distributions.

Often, the Bayesian network is not given but we learn it from data. The process of learning a Bayesian network is twofold: first, we learn the structure and then we learn the parameters of the local distributions. In this paper, we will study learning the structure of a Bayesian network. Specifically, we solve the Bayesian network structure learning (BNSL) problem using the so-called *score-based* approach where each structure is assigned a score based on how well it fits to the data and the goal is to find a structure that maximizes the score.

Exact structure learning algorithms are guaranteed to find an optimal structure, that is, a structure that maximizes the score. Exact structure learning is NP-hard (Chickering, 1996) so it is unlikely that exact algorithms scale-up to large networks in worst case scenarios. Even though sometimes even networks with few dozens of variables are too large for state-of-the-art exact algorithms (Cussens, 2011), they can solve easy instances with up to few hundred variables. When the exact algorithms cannot handle a dataset, one typically resorts to various scalable heuristics (Scanagatta et al., 2015; Tsamardinos et al., 2006). A downside of using heuristics is that they do not provide any guarantees for the quality of the result.

We present a new structure learning algorithm that attempts to bridge the gap between these two extremes. Specifically, we speed up the algorithm by relaxing the requirement of guaranteeing the optimal DAG. At the same time, we are still able to give an estimate of the quality of the found solution.

Our method builds upon GOBNILP (Cussens, 2011), the state-of-the-art exact algorithm, which is based on integer linear programming (ILP). In score-based structure learning, it is common to use so-called decomposable scores which means that the score of a structure is a sum of local scores for node-parent set

pairs. Many algorithms, including GOBNILP, require that these local scores are computed as a preprocessing step and given as an input to an optimization algorithm. One of the challenges using GOBNILP is that a large number of local scores makes solving the integer linear program slower. We tackle this problem by approximating local scores with approximate scores for each potential edge and thereby restricting the size of the input to a quadratic number of scores. Given edge scores, finding the optimal structure reduces to the maximum acyclic subgraph (MAS) problem; hence, we call our method *BNSL2MAS*. Also the MAS problem is NP-hard (Karp, 1972) but in practice it is much faster to solve.

A key challenge in our approach is the quality of the approximation. To enable us to assess the quality of the found network, we learn the approximate scores under the constraint that the approximate score (sum of the edge scores) for each node-parent set pair upper bounds the local score. This guarantees that the approximate score of the optimal solution of the MAS problem gives an upper bound for the score of the optimal solution of the BNSL problem.

The above-described version of BNSL2MAS is fast but it has a considerable weakness: It returns networks that are dense and upper bounds are too high to have any practical value. To get tighter upper bounds, we can add various additional constraints (described in Section 3.3). However, tightening the upper bound does not come for free. It leads to a tradeoff between speed and tightness of the bound. Our empirical results are mixed: We observed that the variants of BNSL2MAS that were fast did not give usable upper bounds. On the other hand, the variants that gave non-trivial upper bounds were slow and sometimes returned graphs which were of poorer quality.

Related work. There is limited amount of "approximate" algorithms for Bayesian network structure learning. Most notably, while integer linear programming (ILP) -based algorithms (Jaakkola et al., 2010; Cussens, 2011) are usually used as exact algorithms, they can be used as anytime algorithms. That is, it is possible to interrupt the algorithm at any time and output the best DAG found so far. Furthermore, ILP solves relaxations and optimal solutions for the relaxations give upper bounds for the score of the optimal DAG.

Other works include an approximation algorithm developed by Ziegler (2008). The algorithm gives k-approximation where k is the maximum size of the parent set. Also greedy equivalence search (GES) (Chickering, 2002) can be seen as a heuristic with quality guarantees. However, the guarantees hold only asymptotically.

Recently, there has also been work on developing anytime algorithms for Bayesian network structure learning (see, e.g., Lee and van Beek (2017); Scanagatta et al. (2017)). These algorithms typically consist of sampling node orders and then finding a best DAG that is compatible with the order.

2. Preliminaries

2.1 Score-based Structure Learning

Let $G = (N, A)$ be a DAG where N is the node set and A is the arc set. We denote the parent set of node v in G by A_v. Let n denote the cardinality of N.

The score of a DAG measures how well the DAG fits to the data. A score is decomposable, if the score of a DAG G can be written as

$$S(G) = \sum_{v \in N} S_v(A_v),$$

where $S_v(A_v)$ is the local score of node v given the parent set A_v. Commonly used scores such as BDe, BDeu, and BIC are decomposable. Our approach works for any decomposable score.

The Bayesian network structure learning (BNSL) problem with a decomposable score can be defined as

$$\arg\max_G \sum_v S_v(A_v)$$

s.t. A_v representing a DAG.

Note that A_v refers to the parent set of v in the graph G.

Without any restrictions, the input of the BNSL problem would be 2^{n-1} local scores for every v. Generally, computing such a number of local scores is not feasible. Thus, one typically restricts the number of local scores. To this end, let $\mathcal{F}_v$ denote the set of potential parent sets of the node v. We treat $S_v(A_v) = -\infty$ for all $A_v \notin \mathcal{F}_v$. One common way to restrict the size of $\mathcal{F}_v$ is pruning, that is, removing or not computing local scores for parent sets that cannot be a part of an optimal DAG. A commonly used pruning rule is based on the observation that if $S_v(W) > S_v(W')$ and $W \subset W'$ then W' cannot be the parent set v in the optimal graph. Another way to restrict the number of local scores is to assume that the maximal size of the parent sets is bounded by a small integer k.

We note that for large data sets, the number of local scores to compute becomes quickly restrictive even for a small k. For example, with 500 nodes and $k = 3$ one would need to compute approximately $500^4 \approx 6 \times 10^{10}$ local scores. Thus, heuristics that scale up to thousands of nodes, such as Scanagatta et al. (2015), compute local scores greedily online. We concentrate on slightly smaller networks and assume that the local scores are given (though we do assume that they have been pre-pruned).

2.2 Integer Linear Programming Formulation

The state of the art in exact score-based structure learning in Bayesian networks is based on integer linear programming (ILP) (Jaakkola et al., 2010; Cussens, 2011). In an integer linear programming formulation, one introduces a set of binary variables. The current state of the art involves the so-called "family variables"; a family variable $I_v(W)$ takes value 1 if W is the parent set of v in the DAG, and 0 otherwise. Now the optimization problem can be written as

$$\arg\max_{I_v(W)} \sum_{v,W} S_v(W) I_v(W)$$

s.t. the variables $I_v(W)$ represent a DAG.

We note that the constraints guaranteeing that the family variables $I_v(W)$ represent a DAG can be formulated in several different ways. GOBNILP uses so-called cluster constraints to guarantee acyclicity[1].

We note that the ILP formulation has one variable for each potential parent set in $\mathcal{F}_v$. The number of ILP variables can affect the optimization speed drastically and be the performance bottleneck in practice when the number of potential parent sets is large. This is a serious challenge because with a large number of nodes we tend to have lots of potential parent sets even when indegrees are small. Next, we will present edge scores that are designed to alleviate this problem.

3. Proposed Method

In this section, we will introduce the BNSL2MAS method for the BNSL problem. As mentioned in Section 2.1, we assume that we are given local scores $S_v(A_v)$ for $v \in N$ and $A_v \in \mathcal{F}_v$ and our goal is to find a DAG G that maximizes the sum of local scores. We do not directly solve the BNSL problem but we first approximate the local scores using edge scores and thereby convert the problem to the maximum acyclic subgraph problem which is easier to solve in practice.

1. These constraints are based on the observation that in a DAG, every subset of nodes has at least one node that does not have any parents in that subset.

3.1 Edge Scores

The core idea of our work is to approximate local scores $S_v(W)$ using a sum of edge scores $e_v(w)$. The rationale behind this approximation is that the resulting ILP formulation has only a quadratic number of variables instead of potentially an exponential number of variables. Therefore, it is expected that finding the DAG that maximizes the approximate score is faster than finding the DAG that maximizes the original score.

Formally, we define an approximate local score for a node v and a parent set W by $\widetilde{S}_v(W) = b_v + \sum_{w \in W} e_v(w)$ where b_v is the bias of v and $e_v(w)$ are the edge scores. This approximate score is additive, that is, each approximate local score is a sum of individual edge score contributions from the parent nodes w with respect to the node v, plus a bias that represents the score estimation of v having no parent at all.

Given local scores $S_v(W)$, we want to estimate the values taken by the variables b_v and $e_v(w)$. To this end, we define a loss function $L\big(S_v(W), \widetilde{S}_v(W)\big)$. In our experiments, we have used the absolute loss: $L\big(S_v(W), \widetilde{S}_v(W)\big) = |S_v(W) - \widetilde{S}_v(W)|$ and the squared loss: $L\big(S_v(W), \widetilde{S}_v(W)\big) = (S_v(W) - \widetilde{S}_v(W))^2$. However, other loss functions are also possible. Furthermore, in order to get approximation guarantees, we constrain the approximate local scores to always upper bound the true local scores.

Now, for each node v its bias and edge scores can be estimated solving

$$\begin{aligned} &\underset{b_v, e_v(w)}{\operatorname{argmin}} \sum_{W \in \mathcal{F}_v} L\big(S_v(W), \widetilde{S}_v(W)\big) \\ &\text{s.t. } S_v(W) \leq \widetilde{S}_v(W) \quad \forall W \in \mathcal{F}_v \end{aligned} \tag{1}$$

where $\widetilde{S}_v(W) = b_v + \sum_{w \in W} e_v(w)$.

The approximation guarantees hold for any loss function. The tightness of the bounds and the time requirements of solving the optimization problem, however, may be affected by the choice of the loss function.

The quadratic loss function results in a quadratic optimization problem which can be solved efficiently with solvers like GUROBI. In practice, even for large and dense datasets, the estimation of approximate scores is very fast compared to solving the structure learning problem itself. This is mainly due to the bias and edge scores being continuous variables, as well as the models for each child node v being independent meaning we can easily parallelize them.

There are at most $n(n-1)$ edge scores. We note that the above formulation yields a non-zero edge score $e_v(w)$ only when w is a member of at least one potential parent set, that is, $e_v(w)$ is non-zero only if $w \in W$ for some $W \in \mathcal{F}_v$. This can further simplify the following structure learning problem, especially when the underlying DAG is sparse.

3.2 Maximum Acyclic subgraph

Given the edge scores, solving BNSL reduces to solving an instance of the maximum acyclic subgraph (MAS) problem which is still NP-hard, as mentioned before. In practice, however, MAS can be solved significantly faster than BNSL.

Formally, the *maximum acyclic subgraph (MAS)* problem is defined as follows. We are given a directed graph $G' = (N, A')$ with a weight $s(a)$ assigned for each arc $a \in A'$. The goal is to find a directed acyclic graph $G = (N, A)$ such that $A \subseteq A'$ with maximum total weight $\sum_{a \in A} s(a)$. We note that solving MAS is equivalent to solving the feedback arc set (FAS) problem where one is given a directed weighted graph and the goal is to remove the lightest set of arcs to make the graph acyclic[2].

We solve MAS using integer linear programming. Our ILP formulation of MAS is as follows: consider the weighted graph whose edges are weighted by the edge scores $e_v(w)$. Now, for each directed edge (w, v)

2. MAS and FAS are complementary: the feedback arc set equals to $A' \setminus A$.

define a binary variable $J_v(w)$ which takes value 1 if w is a parent of v in the DAG, 0 otherwise. Then the optimization problem can be written as

$$\underset{J_v(w)}{\operatorname{argmax}} \sum_{v,w} e_v(w) J_v(w)$$

$$\text{s.t. the } J_v(w) \text{ representing a DAG.}$$

There are several alternative ways to guarantee that the resulting graph is a DAG. We solve MAS using the lazy set cover ILP formulation as in Baharev et al. (2015)[3].

The constraints to guarantee acyclicity can be formulated as follows. Let c be a directed cycle with length $|\mathsf{c}|$. Let $\mathcal{C}$ be the set of cycles in G'. If G is a DAG then at least one arc of each cycle has to be absent from G. Therefore, to ensure that G is a DAG, we can add the following constraints: for each cycle $\mathsf{c} \in \mathcal{C}$ it holds that

$$\sum_{(w,v)\in\mathsf{c}} J_v(w) \leq |\mathsf{c}| - 1.$$

One challenge with this formulation is that the ILP contains one constraint for each cycle in G'. In large or dense graphs there are potentially lots of cycles and it is usually not feasible to explicitly include all of them. In this case, we use the same approach as Cussens (2011) and add constraints lazily as cutting planes. That is, we start with a relaxation and whenever we find a feasible solution for the relaxation, it is checked whether the solution is acyclic. If not, we use the cutting plane approach and add at least one cycle constraint that make the found solution infeasible. In this approach, finding good cutting planes is essential for solving the problem quickly. We use a simple heuristical approach: for each edge in the found graph, we find the shortest cycle that goes through the edge (if the edge is a part of a cycle).

In Section 3.1, we constrained the approximate local scores to upper bound the local scores. Theorem 1 implies that the approximate score of the DAG found by solving the MAS problem upper bounds the (true) score of the DAG found by solving the BNSL problem.

Theorem 1 *Let G_{opt} be the optimal DAG of the BNSL problem and $\widetilde{G}_{opt}$ be the optimal DAG of the MAS problem. Furthermore, let $S_v(W) \leq \widetilde{S}_v(W) \quad \forall v \in N \quad \forall W \in \mathcal{F}_v$. Then,*

$$S(G_{opt}) \leq C + \widetilde{S}(\widetilde{G}_{opt}),$$

where $C = \sum_{v\in N} b_v$ is the sum of bias terms.

Proof For any graph $G = (N, A)$, we have that

$$\begin{aligned} S(G) &= \sum_{v\in N} S_v(A_v) \\ &\leq \sum_{v\in N} \widetilde{S}_v(A_v) \\ &= \sum_{v\in N} \left(b_v + \textstyle\sum_{w\in A_v} e_v(w) \right) \\ &= C + \sum_{v\in N} \textstyle\sum_{w\in A_v} e_v(w) \\ &= C + \widetilde{S}(G). \end{aligned}$$

3. Baharev et al. (2015) give their formulation for FAS problem but we adapt it for the MAS problem.

Because $\widetilde{G}_{opt}$ maximizes the approximate score $\widetilde{S}$, it follows that $\widetilde{S}(G_{opt}) \leq \widetilde{S}(\widetilde{G}_{opt})$. Finally, because the approximate score upper bounds the local score, we get the upper bound guarantee

$$S(G_{opt}) \leq C + \widetilde{S}(\widetilde{G}_{opt}).$$

■

3.3 Tightening the Upper Bound

One challenge with the MAS formulation above is that the obtained networks tend to be dense. This happens because the edge scores do not penalize complexity and thus MAS keeps adding edges with positive weights as long as they do not induce cycles.

This denseness has two negative consequences. First, the found network $\widetilde{G}_{opt}$, which is optimal with respect to approximate scores $\widetilde{S}$, may not have a high score with respect to the true scores S. Second, the upper bound depends on the weights of the included edges, adding an extra edge makes the bound looser. As a result, the upper bounds provided by the naive formulation above are usually not very useful.

Fortunately, we can alleviate these problems. First, we note that if the parent set of v in $\widetilde{G}_{opt}$ is not among the candidate parent sets, that is, $A_v \notin \mathcal{F}_v$, we can improve the score by replacing A_v by its highest-scoring subset with respect to true local scores, that is, $\arg\max_{A'_v \subset A_v} S_v(A'_v)$. In other words, given $\widetilde{G}_{opt} = (N, A)$ we can construct a graph $\widetilde{G}'_{opt} = (N, A')$ such that $\widetilde{G}'_{opt}$ maximizes the score $S(\widetilde{G}'_{opt})$ subject to $A' \subseteq A$. Because removing edges cannot make an acyclic graph cyclic, we can select the parent set of each node independently and therefore the running time is proportional to the size of $\mathcal{F}_v$. We refer to this variant of the BNSL2MAS method as *BNSL2MAS (Base)*.

Second, we can tighten the upper bound by adding additional constraints. A simple way is to bound the maximum indegree of $\widetilde{G}_{opt}$. If the size of the largest parent set in $\mathcal{F}_v$ is k then we can add constraints

$$\sum_{w \in N \setminus \{v\}} J_v(w) \leq k$$

for all v. A weakness of this approach is that it is still possible that the parent set of v in $\widetilde{G}_{opt}$ is not among the candidate parent sets: $A_v \notin \mathcal{F}_v$. We refer to this variant as *BNSL2MAS (Bounded parent set)*.

A more sophisticated approach is to restrict the solutions of the MAS problem to be DAGs whose parent sets are among candidate parents sets. That is, A_v can be the parent set of v in $\widetilde{G}_{opt}$ only if $A_v \in \mathcal{F}_v$. To achieve this, we can add an additional binary variable $K_v(W)$ for all $W \in \mathcal{F}_v$. The variable $K_v(W)$ equals to 1 if and only if W is the parent set of v. This can be achieved by adding the following constraints:

$$\begin{aligned} \sum_{w \in W} J_v(w) &\geq |W| K_v(W), \\ \sum_{w \notin W} J_v(w) &\leq n - n K_v(W). \end{aligned}$$

Then, we add a constraint that each node has exactly one parent set, that is, $\sum_{W \in \mathcal{F}_v} K_v(W) = 1$. We refer to the variant of BNSL2MAS that includes all the candidate parent set constraints as *BNSL2MAS (Complete candidate parent)*.

Another way to restrict the parent sets to candidate parent sets is to add the following constraints: $\forall v \in N$ and $\forall W \notin F_v$ it is required

$$\sum_{w \in W} J_v(w) \leq \sum_{w' \in \hat{W} \setminus W} J_v(w') + |W| - 1,$$

Name	n	Local scores	Name	n	Local scores
autos	26	25,238	letter	17	18,841
carpo_100	60	5,068	mushroom	23	13,025
carpo_10000	60	16,391	Pigs_10000	441	304,219
Diabetes_1000	413	21,493	wdbc_Bde	31	13,473
Diabetes_10000	413	262,129	wdbc_BIC	31	14,613

Table 1: Data sets with the number of nodes n and the number of local scores

where

$$\hat{W} = \bigcup_{W' \in F_v : W' \supset W} W'.$$

Intuitively, this means that if the parent set of v contains a set $W \notin \mathcal{F}_v$ then the parent set must include at least one additional node. The additional node must be a member of some candidate parent set that is a superset of W. We note $\hat{W} = N \setminus W$ would be also a valid constraint but the above mentioned constraint is tighter. The previous formulation is not completely unproblematic: We need to have one constraint for every $W \notin \mathcal{F}_v$ which means that usually we will have exponential number of constraints. Thus, adding all of them is typically not possible. Fortunately, we can again add the constraints as cutting planes during the optimization process. Whenever we find a graph G with $A_v \notin \mathcal{F}_v$, we can add the corresponding constraint. We refer to the variant that adds candidate parent set constraints lazily as cutting planes as *BNSL2MAS (Lazy candidate parent)*.

It should be noted that removing the constraint $S_v(W) \leq \widetilde{S}_v(W)$ in Equation 1 that forces the approximate score to be an upper bound for the corresponding true score would make the loss smaller and thereby the approximation would be tighter. However, the constraint is essential for the quality guarantees and after removing it we could not guarantee that the solution of the MAS problem would upper bound the score of the optimal solution of the BNSL problem.

4. Experiments

4.1 Test Setup

4.1.1 Implementation

The method was implemented using Python. We used Gurobi (version 9.0) to solve linear, quadratic, and integer linear programs.

4.1.2 Data sets

We use a subset of the data sets listed in the GOBNILP home page[4]. As we are interested in scalability of our method, we restrict our analysis to the most challenging data sets. To this end, we selected data sets for which GOBNILP used at least 100 seconds (or GOBNILP was not able to find the optimal network at all). The selected data sets can be seen in Table 1.

We used the same scoring criteria that were used in the original data sets, that is, either BDe or BIC. For wdbc, we have two versions, one for each scoring criterion.

4. https://www.cs.york.ac.uk/aig/sw/gobnilp/

4.1.3 EXPERIMENTS

In Section 3, we proposed several different variants. The goal of this experiment is to investigate which of the variants of the proposed method work well. We consider the following variants.

For computing edge scores, we compare two loss functions: absolute loss and squared loss.

For different variants of BNSL2MAS, we consider BNSL2MAS (Base), BNSL2MAS (Bounded parent sets), BNSL2MAS (Complete candidate parents), BNSL2MAS (Lazy candidate parents). As mentioned in Section 3.2, we can solve one of the two equivalent optimization problems: maximum acyclic subgraph (MAS) or feedback arc set (FAS). Both formulations give the same result (if we run them long enough to the optimality) but the running time can be different. Therefore, if we run the algorithms for a fixed time and at least one of them does not finish then we can also get different graphs. Thus, we compare both MAS and FAS formulations.

Performance measures. In all of the experiments, we measured the performance of the methods by scores of found DAGs and running time. For ILP-based methods, we also recorded upper bounds. For the proposed method, we computed both the approximate score $\widetilde{S}$ and the "true" score S for each DAG.

All methods had a time limit of one hour of running time. We evaluated methods in anytime fashion: Whenever a method found a DAG that has higher score than any of the previously found DAGs, we recorded the score and time of the newly found DAG. Furthermore, all algorithms were given pre-computed local scores[5] as an input and computing the local scores did not count towards the running time except for hill climbing, for which raw datasets were used.

Benchmarks. We benchmark the proposed method against both an exact algorithm and two heuristic algorithms. As the exact benchmark, we use GOBNILP (Bartlett and Cussens, 2017) which can scale up to networks with few hundred variables if the network is sparse.

As another benchmark we use the greedy hill climbing. In greedy hill climbing, there are three operators that are used to improve the DAG: adding a new edge, removing an existing edge or flipping an edge. The greedy hill climbing starts with an empty DAG and greedily applies the operator that increases the score most. This is continued until none of the operators increases the score of the DAG. We benchmarked hill climbing from bnlearn (Scutari, 2010), an optimized implementation that uses score caching for maximized efficiency.

Our last benchmark is WINASOBS (Scanagatta et al., 2017), where an ordering-based search algorithm is coupled with an iterated local search meta-heuristic that uses a custom window insertion operator. WINASOBS currently yields state-of-the-art performance on large graphs with several thousands of nodes.

Computer. Tests were conducted using a computer with Intel(R) Core(TM) i5-7500 processor with four 3.40GHz cores. The computer had 32Gb RAM and its operating system was Ubuntu 18.04.4 LTS (64bit).

4.2 Results

4.2.1 COMPUTING EDGE SCORES

We computed edge scores with both squared and absolute loss. We observed that computing the edge scores is very fast. In the case of squared loss, for all but two of our data sets computing edge scores took less than 2 seconds in total. The two most time-consuming data sets were Diabetes_10000 and Pigs_10000 for which computing the edge scores took 35 and 72 seconds, respectively. Computing edge scores with absolute loss is even faster: computing edge scores for any data set took always less than 2 seconds.

4.2.2 BNSL2MAS

Next, we will review some of the experimental results. Due to space constraints, we show only part of the results. Let us start by analysing the BNSL2MAS (Base). Results from selected data sets are shown in

5. Loaded from `https://www.cs.york.ac.uk/aig/sw/gobnilp/data/`.

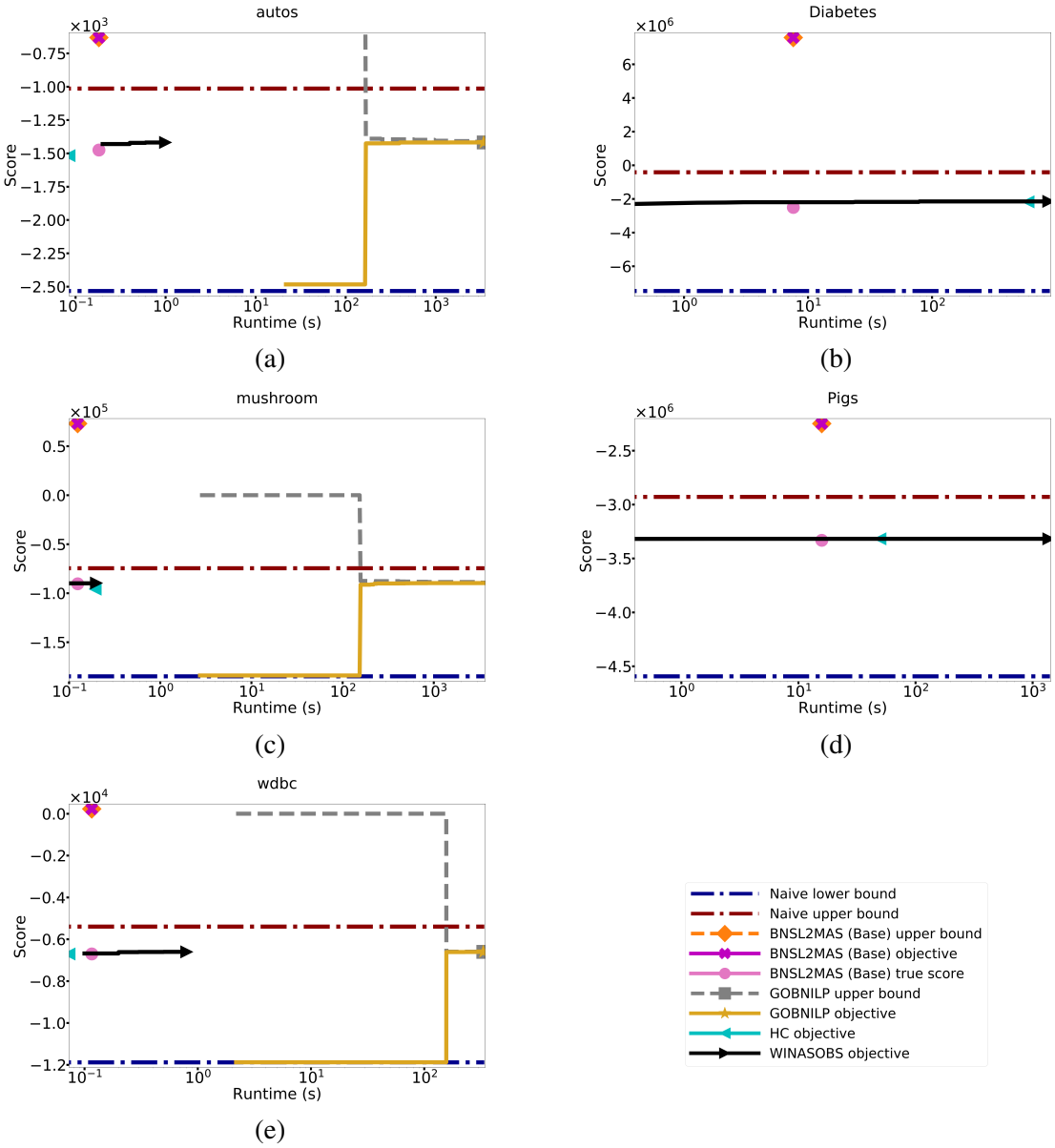

Figure 1: Scores and running times for BNSL2MAS (Base) and BNSL2MAS (Lazy candidate parents) with the benchmarks. Scores are the score of the best DAG found at a given data point. Edge scores were computed using squared loss. Marker at the end of a curve denotes that the method has finished (otherwise, the execution was stopped at the time limit). Data sets: (a) autos, (b) Diabetes_10000, (c) mushroom, (d) Pigs_10000, and (e) wdbc.

Figure 1. To help us to put the scores and bounds in perspective, we compare them to naive bounds. As a *naive lower bound* we use the score of the empty graph. To get a *naive upper bound* for the optimal DAG, we select the highest scoring parent set for each node and compute the sum. Clearly, there cannot be a DAG that has a higher score. Furthermore, we show the hill climbing algorithm, WINASOBS, and GOBNILP as

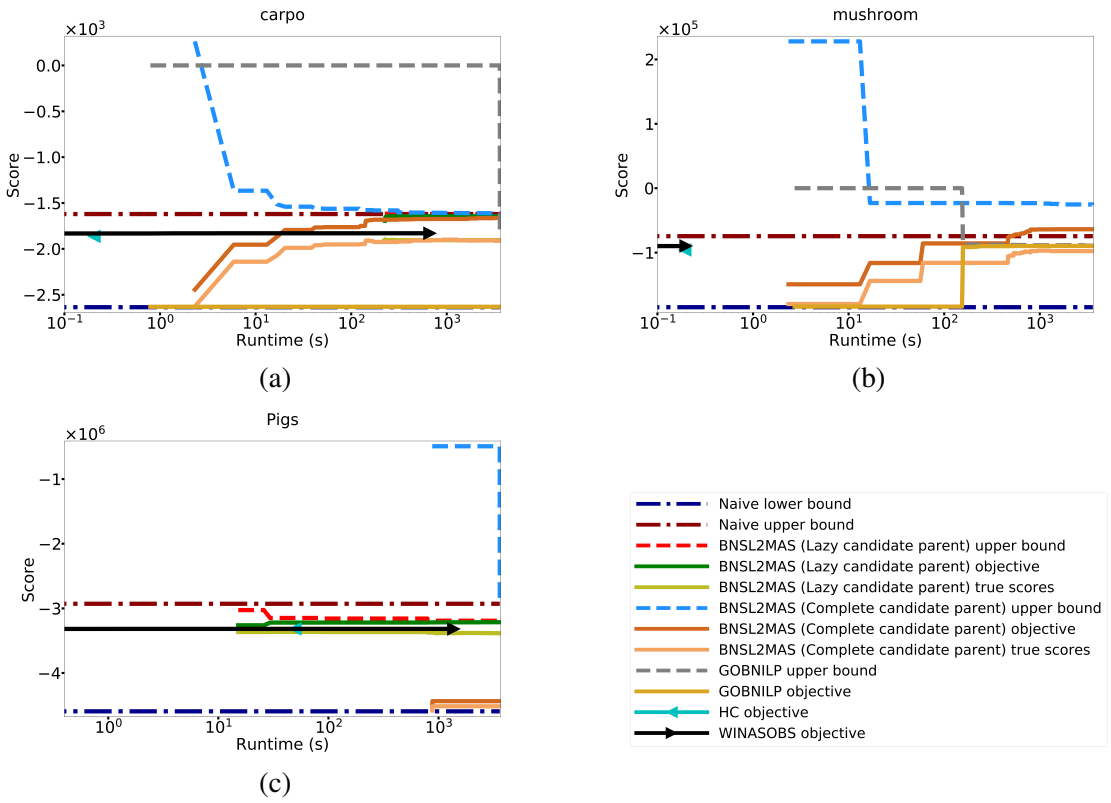

Figure 2: Score and running times for BNSL2MAS (Complete candidate parents) and BNSL2MAS (Lazy candidate parent) with benchmarks. Scores are the score of the best DAG found at a given data point. Edge scores were computed using squared loss. Marker at the end of a curve denotes that the method has finished (otherwise, the execution was stopped at the time limit). Datasets: (a) carpo_100, (b) mushroom, and (c) Pigs_10000.

benchmarks. In the plots, whenever a curve for some method is missing that means that the method did not find any feasible solutions before the time limit.

Our first observation is that BNSL2MAS (Base) is very fast. With the absolute loss, the slowest cases were the networks with over 400 nodes. Solving the MAS problem took less than 11 seconds for Diabetes_10000 and less than 25 seconds for Pigs_10000. With squared loss, the algorithm was even faster: running times for Diabetes_10000 and Pigs_10000 were less than 3 seconds and 13 seconds, respectively.

If we take a look at the DAGs found, we can see that, indeed, as mentioned in Section 3.3, solving BNSL2MAS without additional constraints for parent sets leads to dense networks. For example, the mushroom data set which consist of 23 nodes. The DAG found has 228 edges (average in-degree is 9.9) and maximum in-degree is 20. A graph with 23 nodes can have at most 253 edges. Thus, the found graph is almost complete. In comparison, the optimal graph found by GOBNILP has only 67 edges and maximum in-degree 4. As each edge that is added has a positive score, the upper bound is always much higher than the naive upper bound. Thus, in this case we cannot really claim that the base version has quality guarantees.

To evaluate the quality of the DAGs found by BNSL2MAS (Base), we selected the highest scoring candidate parent set for each node among the parents in the found graph and computed the true score of this DAG. The base version seems to perform roughly as well as the hill climbing heuristic: With squared

loss, the base version found a higher scoring network 4 times out of 10. WINASOBS found a higher-scoring DAG for all of the data sets. WINASOBS also finds good solutions very fast. We observe that the curves for WINASOBS in Figure 1 are almost horizontal indicating that there is no significant improvement after the initial phase.

To get tighter bounds, we can use either bounded parent set constraints or candidate parent set constraints. Selected results comparing BNSL2MAS (Complete candidate parent) and BNSL2MAS (Lazy candidate parent) are shown in Figure 2. The results are mixed. Still sometimes the upper bound is higher than the naive upper bound. Both of the versions are significantly slower than BNSL2MAS (Base). However, neither of them consistently overperforms the other. As a rule of thumb, BNSL2MAS (Complete candidate parent) seems to perform better on small data sets and significantly slow down with larger data sets; this is natural because it has one constraint for every existing node-parent set pair in the local scores. Furthermore, sometimes BNS2MAS (Complete candidate parent) and BNSL2MAS (Lazy candidate parent) are faster than GOBNILP and sometimes they are slower. We also note that in many cases BNSL2MAS (Base) finds better solutions than both BNSL2MAS (Complete candidate parent) and BNSL2MAS (Lazy candidate parent).

Due to space constraints, we do not show any plots about the performance of BNSL2MAS (Bounded parent sets). However, in general BNSL2MAS (Bounded parent sets) ends up somewhere in the middle of BNSL2MAS (Base) and the candidate parent set variants with respect to speed and loses to BNSL2MAS (Base) almost always with respect to the score. We also compared MAS and FAS version of BNSL2MAS (Plots not shown). There were no systematic differences, one was faster than the other for about half of the time. We also note that the differences between MAS and FAS were reasonably small.

5. Discussion

On the positive side, we observed that approximating local scores with edge scores and converting BNSL to MAS can significantly speed up learning Bayesian network structures. BNSL2MAS (Base) is very fast and could work with even larger data sets than what were considered in this paper. However, in practice the base version behaves like an order-finding heuristic and does not have quality guarantees.

Our empirical results are somewhat disappointing with respect to quality guarantees. The main result is that developing a scalable method with good quality guarantees seems to be a difficult task. There is a clear tradeoff: Adding seemingly simple constraints to tighten the upper bounds makes the optimization dramatically more difficult increasing the running time by several orders of magnitude even though the size of the search space decreases. We also observed that having quality guarantees can actually decrease the quality of the found solutions! The observation that additional constraints can significantly slow down ILP is not unique. For example, ILP-based methods of learning bounded tree-width Bayesian networks (Parviainen et al., 2014; Scanagatta et al., 2016) are orders of magnitude slower than GOBNILP even though bounded tree-width Bayesian networks are a small subset of all Bayesian networks. These results seem to suggest that the most practical version of BNSL2MAS is BNSL2MAS (Base).

We also note that the state-of-the-art anytime algorithm WINASOBS clearly outperformed BNSL2MAS. The main weakness of BNSL2MAS seems to be that the additive approximation is rather limiting. Thus, making BNSL2MAS really competitive would require finding a less restrictive approximate score without significantly increasing running time.

Acknowledgments

We thank James Cussens for fruitful conversations. The authors are affiliated with the CEDAS center in Bergen.

References

A. Baharev, H. Schichl, and A. Neumaier. An exact method for the minimum feedback arc set problem. 2015.

M. Bartlett and J. Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

D. M. Chickering. *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer-Verlag, learning from data: artificial intelligence and statistics v edition, January 1996.

D. M. Chickering. Optimal Structure Identification With Greedy Search. *Journal of Machine Learning Reseach*, 3:507–554, 2002.

J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 153–160, 2011.

T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian network structure using lp relaxations. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 358–365, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

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.

P. Parviainen, H. S. Farahani, and J. Lagergren. Learning Bounded Tree-width Bayesian Networks using Integer Linear Programming. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 751–759, 2014.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7.

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: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1864–1872, 2015.

M. Scanagatta, G. Corani, C. P. de Campos, and M. Zaffalon. Learning treewidth-bounded Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems 29*, pages 1462–1470. 2016.

M. Scanagatta, G. Corani, and M. Zaffalon. Improved local search in Bayesian networks structure learning. In *Proceedings of The 3rd International Workshop on Advanced Methodologies for Bayesian Networks*, volume 73 of *Proceedings of Machine Learning Research*, pages 45–56, 2017.

M. Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3): 1–22, 2010.

I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 2006.

V. Ziegler. Approximation algorithms for restricted Bayesian network structures. *Information Processing Letters*, 108(2):60–63, Sept. 2008.