Journal of Machine Learning Research 18 (2017) 1-32 Submitted 1/17; Revised 8/17; Published 10/17

Bayesian Network Learning via Topological Order

Young Woong Park YWPARK@IASTATE.EDU
College of Business
Iowa State University
Ames, IA 50011, USA

Diego Klabjan D-KLABJAN@NORTHWESTERN.EDU
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL 60208, USA

Editor: Zhihua Zhang

Abstract

We propose a mixed integer programming (MIP) model and iterative algorithms based on topological orders to solve optimization problems with acyclic constraints on a directed graph. The proposed MIP model has a significantly lower number of constraints compared to popular MIP models based on cycle elimination constraints and triangular inequalities. The proposed iterative algorithms use gradient descent and iterative reordering approaches, respectively, for searching topological orders. A computational experiment is presented for the Gaussian Bayesian network learning problem, an optimization problem minimizing the sum of squared errors of regression models with L1 penalty over a feature network with application of gene network inference in bioinformatics.

Keywords: Bayesian networks, topological orders, Gaussian Bayesian network, directed acyclic graphs

1. Introduction

Directed graph G is a directed acyclic graph (DAG) or acyclic digraph if G does not contain a directed cycle. In this paper, we consider a generic optimization problem over a directed graph with acyclic constraints, which require the selected subgraph to be a DAG.

Let us consider a complete digraph G. Let m be the number of nodes in digraph G, $Y \in \mathbb{R}^{m\times m}$ a decision variable matrix associated with the arcs, where Y_{jk} is related to arc (j,k), $supp(Y) \in \{0,1\}^{m\times m}$ the 0-1 (adjacency) matrix with $supp(Y)_{jk} = 1$ if $Y_{jk} \neq 0$, $supp(Y)_{jk} = 0$ otherwise, $G(supp(Y))$ the sub-graph of G defined by $supp(Y)$, and let $\mathcal{A}$ be the collection of all acyclic subgraphs of G. Then, we can write the optimization problem with acyclic constraints as

$$\min_{Y} \; F(Y) \quad s.t. \quad G(supp(Y)) \in \mathcal{A}, \tag{1}$$

where F is a function of Y.

Acyclic constraints (or DAG constraints) appear in many network structured problems. The maximum acyclic subgraph problem (MAS) is to find a subgraph of G with maximum cardinality while the subgraph satisfies acyclic constraints. MAS can be written in the

form of (1) with $F(Y) = -\|supp(Y)\|_0$. Although exact algorithms were proposed for a superclass of cubic graphs (Fernau and Raible, 2008) and for general directed graphs (Kaas, 1981), most of the works have focused on approximations (Even et al., 1998; Hassin and Rubinstein, 1994) or inapproximability (Guruswami et al., 2008) of either MAS or the minimum feedback arc set problem (FAS). FAS of a directed graph G is a subgraph of G that creates a DAG when the arcs in the feedback arc set are removed from G. Note that MAS is closely related to FAS and is dual to the minimum FAS. Finding a feedback arc set with minimum cardinality is $\mathcal{NP}$-complete in general (Karp, 1972). However, minimum FAS is solvable in polynomial time for some special graphs such as planar graphs (Lucchesi and Younger, 1978) and reducible flow graphs (Ramachandran, 1988), and a polynomial time approximation scheme was developed for a special case of minimum FAS, where exactly one arc exists between any two nodes, called tournament (Kenyon-Mathieu and Schudy, 2007).

DAGs are also extensively studied in Bayesian network learning. Given observational data with m features, the goal is to find the true unknown underlying network of the nodes (features) while the selected arcs (dependency relationship between features) do not create a cycle. In the literature, approaches are classified into three categories: (i) score-based approaches that try to optimize a score function defined to measure fitness, (ii) constraint-based approaches that test conditional independence to check existence of arcs between nodes (iii) and hybrid approaches that use both constraint and score-based approaches. Although there are many approaches based on the constraint-based or hybrid approaches, our focus is solving (1) by means of score-based approaches. For a detailed discussion of constraint-based and hybrid approaches and models for undirected graphs, the reader is referred to Aragam and Zhou (2015) and Han et al. (2016).

For estimating the true network structure by a score-based approach, various functions have been used as different functions give different solutions and behave differently. Many works focus on penalized least squares, where penalty is used to obtain sparse solutions. Popular choices of the penalty term include BIC (Lam and Bacchus, 1994), L_0-penalty (Chickering, 2002; Van de Geer and Bühlmann, 2013), L_1-penalty (Han et al., 2016), and concave penalty (Aragam and Zhou, 2015). Lam and Bacchus (1994) use minimum-description length as a score function, which is equivalent to BIC. Chickering (2002) proposes a two-phase greedy algorithm, called greedy equivalence search, with the L_1 norm penalty. Van de Geer and Bühlmann (2013) study the properties of the L_0 norm penalty and show positive aspects of using L_0 regularization. Raskutti and Uhler Raskutti and Uhler (2013) use a variant of the L_0 norm. They use cardinality of the selected subgraph as the score function where the subgraphs not satisfying the Markov assumption are penalized with a very large penalty. Aragam and Zhou (2015) introduce a generalized penalty, which includes the concave penalty, and develop a coordinate descent algorithm. Han et al. (2016) use the L_1 norm penalty and propose a Tabu search based greedy algorithm for reduced arc sets by neighborhood selection in the pre-processing step.

With any choice of a score function, optimizing the score function is computationally challenging, because the number of possible DAGs of G grows super-exponentially in the number of nodes m and learning Bayesian networks is also shown to be $\mathcal{NP}$-complete (Chickering, 1996). Many heuristic algorithms have been developed based on greedy hill climbing (Chickering, 2002; Heckerman et al., 1995; Han et al., 2016) or coordinate descent (Fu and Zhou, 2013), or enumeration (Raskutti and Uhler, 2013) when the score function

itself is the main focus. There also exist exact solution approaches based on mathematical programming. One of the natural approaches is based on cycle prevention constraints, which are reviewed in Section 2. The model is covered in Han et al. (2016) as a benchmark for their algorithm, but the MIP based approach does not scale; computational time increases drastically as data size increases and the underlying algorithm cannot solve larger instances. Baharev et al. (2015) studied MIP models for minimum FAS based on triangle inequalities and set covering models. Several works have been focused on the polyhedral study of the acyclic subgraph polytopes (Bolotashvili et al., 1999; Goemans and Hall, 1996; Grötschel et al., 1985; Leung and Lee, 1994). In general, MIP models have gotten relatively less attention due to the scalability issue.

In this paper, we propose an MIP model and iterative algorithms based on the following well-known property of DAGs (Cook et al., 1998).

Property 1 *A directed graph is a DAG if and only if it has a topological order.*

A *topological order* or *topological sort* of a DAG is a linear ordering of all of the nodes in the graph such that the graph contains arc (u, v) if and only if u appears before v in the order (Cormen et al., 2009). Suppose that Z is the adjacency matrix of an acyclic graph. Then, by sorting the nodes of acyclic graph $G(Z)$ based on the topological order, we can create a lower triangular matrix from Z, where row and column indices of the lower triangular matrix are in the topological order. Then, any arc in the lower triangular matrix can be used without creating a cycle. By considering all arcs in the lower triangular matrix, we can optimize F in (1) without worrying to create a cycle. This is an advantage compared to arc-based search, where acyclicity needs to be examined whenever an arc is added. Although the search space of topological orders is very large, a smart search strategy for a topological order may lead to a better algorithm than the existing arc-based search methods. Node orderings are used for Bayesian Network learnings based on Markov chain Monte Carlo methods (Ellis and Wong, 2008; Friedman and Koller, 2003; Niinimäki et al., 2016) as alternatives to network structure-based approaches.

The proposed MIP assigns node orders to all nodes and add constraints to satisfy Property 1. The iterative algorithms search over the topological order space by moving from one topological order to another order. The first algorithm uses the gradient to find a better topological order and the second algorithm uses historical choice of arcs to define the score of the nodes.

With the proposed MIP model and algorithms for (1), we consider a Gaussian Bayesian network learning problem with L_1 penalty for sparsity, which is discussed in detail in Section 4. Out of many possible models in the literature, we pick the L_1-penalized least square model from recently published work of Han et al. (2016), which solves the problem using a Tabu search based greedy algorithm. The algorithm is one of the latest algorithms based on arc search and is shown to be scalable when m is large. Further, their score function, L_1 penalized least squares, is convex and can be solved by standard mathematical optimization packages. Hence, we select the score function from Han et al. (2016) and use their algorithm as a benchmark. In the computational experiment, we compare the performance of the proposed MIP model and algorithms against the algorithm in Han et al. (2016) and other available MIP models for synthetic and real instances.

Our contributions are summarized in the following.

1. We consider a general optimization problem with acyclic constraints and propose an MIP model and iterative algorithms for the problem based on the notion of topological orders.
2. The proposed MIP model has significantly fewer constraints than the other MIP models in the literature, while maintaining the same order of the number of variables. The computational experiment shows that the proposed MIP model outperforms the other MIP models when the subgraph is sparse.
3. The iterative algorithms based on topological orders outperform when the subgraph is dense. They are more scalable than the benchmark algorithm of Han et al. (2016) when the subgraph is dense.

In Section 2, we present the new MIP model along with two MIP models in the literature. In Section 3, we present two iterative algorithms based on different search strategies for topological orders. The Gaussian Bayesian network learning problem with L_1-penalized least square is introduced and computational experiment are presented in Sections 4 and 5, respectively.

In the rest of the paper, we use the following notation.

$J = \{1, \cdots, m\}$ = index set of the nodes
$J^k = J \setminus \{k\}$ = index set of the nodes excluding node k, $k \in J$
$Z = supp(Y) \in \{0,1\}^{m \times m}$
π = topological order

Given π, we define $\pi_k = q$ to denote that the order of node k is q. For example, given three nodes a, b, c, and topological order $b - c - a$, we have $\pi_a = 3$, $\pi_b = 1$, and $\pi_c = 2$. With this notation, if $\pi_j > \pi_k$, then we can add an arc from j to k.

2. MIP Formulations based on Topological Order

In this section, we present three MIP models for (1). The first and second models, denoted as MIPcp and MIPin, respectively, are models in the literature for similar problems with acyclic constraints. The third model, denoted as MIPto, is the new model we propose based on Property 1.

A popular mathematical programming based approach for solving (1) is the cutting plane algorithm, which is well-known for the traveling salesman problem formulation. Let $\mathcal{C}$ be the set of all possible cycles and $C_l \in \mathcal{C}$ be the set of the arcs defining a cycle. Let $H(supp(Y), C_l)$ be a function that counts the number of selected arcs in $supp(Y)$ from C_l. Then, (1) can be solved by

$$\textbf{MIPcp} \quad \min_{Y} \; F(Y) \quad s.t. \quad H(supp(Y), C_l) \leq |C_l| - 1, C_l \in \mathcal{C}, \tag{2}$$

which can be formulated as an MIP. Note that (2) has exponentially many constraints due to the cardinality of $\mathcal{C}$. Therefore, it is not practical to pass all cycles in $\mathcal{C}$ to a solver. Instead, the cutting plane algorithm starts with an empty active cycle set $\mathcal{C}^A$ and iteratively adds cycles to $\mathcal{C}^A$. That is, the algorithm iteratively solves

$$\min_{Y} \; F(Y) \quad s.t. \quad H(supp(Y), C_l) \leq |C_l| - 1, C_l \in \mathcal{C}^A, \tag{3}$$

with the current active set $\mathcal{C}^A$, detects cycles from the solution, and adds the cycles to $\mathcal{C}^A$. The algorithm terminates when there is no cycle detected from the solution of (3). One of the drawbacks of the cutting plane algorithm based on (3) is that in the worst case we can add all exponentially many constraints. In fact, Han et al. (2016) study the same model and concluded that the cutting plane algorithm does not scale.

Baharev et al. (2015) recently presented MIP models for the minimum feedback arc set problem based on linear ordering and triangular inequalities, where the acyclic constraints presented were previously used for cutting plane algorithms for the linear ordering problem (Grötschel et al., 1984; Mitchell and Borchers, 2000). For any F, we can write the following MIP model based on triangular inequalities presented in Baharev et al. (2015), Grötschel et al. (1984), and Mitchell and Borchers (2000).

$$
\begin{aligned}
\textbf{MIPin} \quad \min \quad & F(Y) && && \text{(4a)}\\
s.t. \quad & Z = supp(Y), && && \text{(4b)}\\
& Z_{qj} + Z_{jk} - Z_{qk} \leq 1, && 1 \leq q < j < k \leq m, && \text{(4c)}\\
& -Z_{qj} - Z_{jk} + Z_{qk} \leq 0, && 1 \leq q < j < k \leq m, && \text{(4d)}\\
& Z_{jk} \in \{0,1\}, && 1 \leq j < k \leq m && \text{(4e)}
\end{aligned}
$$

Note that Z_{jk} is not defined for all $j \in J^k$ and $k \in J$. Instead of having a full matrix of binary variables, the formulation only uses lower triangle of the matrix using the fact that $Z_{jk} + Z_{kj} = 1$. We can also use this technique to any of the MIP models presented in this paper. However, for ease of explanation, we will use the full matrix, while the computational experiment is done with the reduced number of binary variables. Therefore, the cutting plane algorithm with MIPcp should be more scalable than the implementation in Han et al. (2016), which has twice more binary variables.

Baharev et al. (2015) also provides a set covering based MIP formulation. The idea is similar to MIPcp. In the set covering formulation, each row and column represents a cycle and an arc, respectively. Similar to MIPcp, existence of exponentially many cycles is a drawback of the formulation and Baharev et al. (2015) use the cutting plane algorithm.

Next, we propose an MIP model based on Property 1. Although MIPin uses significantly less constraints than MIPcp, MIPin still has $O(m^3)$ constraints which grows rapidly in m. On the other hand, the MIP model we propose has $O(m^2)$ variables and $O(m^2)$ constraints. In addition to Z, let us define decision variable matrix $O \in \{0,1\}^{m \times m}$.

$$
O_{kq} = \begin{cases} 1 & \text{if } \pi_k = q, \\ 0 & \text{otherwise,} \end{cases} \qquad k \in J, q \in J
$$

Then, we have the following MIP model.

$$\textbf{MIPto} \quad \min \quad F(Y) \tag{5a}$$
$$s.t. \quad Z = supp(Y), \tag{5b}$$
$$Z_{jk} - mZ_{kj} \leq \sum_{r \in J} r(O_{kr} - O_{jr}), \qquad j \in J^k, k \in J, \tag{5c}$$
$$Z_{jk} + Z_{kj} \leq 1, \qquad j \in J^k, k \in J, \tag{5d}$$
$$\sum_{q \in J^k} O_{kq} = 1, \qquad k \in J, \tag{5e}$$
$$\sum_{k \in J^q} O_{kq} = 1, \qquad q \in J, \tag{5f}$$
$$Z, O \in \{0,1\}^{m \times m}, Y \text{ unrestricted} \tag{5g}$$

The key constraint in (5) is (5c). Recall that Z_{jk} indicates which node comes first in the topological order and O_{kr} stores the exact location in the order. With these definitions, (5c) forces correct values of Z_{jk} and Z_{kj} by comparing the order difference. Recall that we can reduce the number of binary variables Z_{jk}'s by plugging $Z_{jk} + Z_{kj} = 1$, but we keep the full matrix notation for ease of explanation. We next show that (5) correctly solves (1).

Proposition 1 *An optimal solution to* (5) *is an optimal solution to* (1).

Proof By Property 1, any DAG has a corresponding topological order. Let π^* be the topological order defined by an optimal solution Y^* for (1). Note that (5e) and (5f) define a topological order. Hence, it suffices to show that (5) gives a DAG given π^*. Note that the right hand side of (5c) measures the difference in the topological order between nodes j and k. If the value is positive, it implies $\pi_k > \pi_j$. Consider (5c) for j_1 and j_2 with $\pi^*_{j_2} > \pi^*_{j_1}$. When $j = j_1$ and $k = j_2$, we have $\sum_{r \in J} r(O_{j_2 r} - O_{j_1 r}) > 0$ in (5c) and at most one of $z_{j_1 j_2}$ and $z_{j_2 j_1}$ can be 1 by (5d). When $j = j_2$ and $k = j_1$, we have $\sum_{r \in J} r(O_{j_1 r} - O_{j_2 r}) < 0$ in (5c) and we must have $z_{j_1 j_2} = 1$ by the left hand side of (5c). Therefore, we have correct value $z_{j_1 j_2} = 1$ when $\pi^*_{j_2} > \pi^*_{j_1}$. This completes the proof. ■

In Table 1, we compare the MIP models introduced in this section. Although all three MIP models have $O(m^2)$ binary variables, MIPto has more binary variables than MIPcp and MIPin due to O_{kq}'s. MIP models MIPin and MIPto have polynomially many constraints, whereas MIPcp has exponentially many constraints. MIPto has the smallest number of constraints among the three MIP models. In the computational experiment, we use a variation of the cutting plane algorithm for MIPcp as it has exponentially many constraints. For MIPin and MIPto, we do not use a cutting plane algorithm.

3. Algorithms based on Topological Order

Although the MIP models introduced in Section 2 guarantee optimality, the execution time for solving an integer programming problem can be exponential in problem size. Further, the execution time could increase drastically in m, as all of the models require at least

Name	Reference	# binary variables	# constraints
MIPcp	(2)	$O(m^2)$	exponential
MIPin	(4)	$O(m^2)$	$O(m^3)$
MIPto	(5)	$O(m^2)$	$O(m^2)$

Table 1: Number of binary variables and constraints of MIP models

$O(m^2)$ binary variables and $O(m^2)$ constraints. In order to deal with larger graphs, we propose iterative algorithms for (1) based on Property 1. Observe that, if we are given a topological order of the nodes, then Z and O are automatically determined in (5). In other words, we can easily obtain a subset of the arcs such that all of the arcs can be used without creating a cycle. Let $\bar{R}$ be the determined adjacency matrix given topological order $\bar{\pi}$. In detail, we set

$$\bar{R}_{jk} = 1 \text{ if } \bar{\pi}_j > \bar{\pi}_k, \ \bar{R}_{jk} = 0 \text{ otherwise.}$$

Let $adj(\bar{\pi})$ be the function generating $\bar{R}$ given input topological order $\bar{\pi}$. If we are given $\bar{\pi}$, then we can generate $\bar{R}$ by $adj(\bar{\pi})$, and solving (1) can be written as

$$\min_{Y} \ F(Y) \quad s.t. \quad \bar{R} \geq supp(Y). \tag{6}$$

Note that (6) has acyclic constraint $\bar{R} \geq supp(Y)$, not $\bar{R} = supp(Y)$. The inequality is needed when we try to obtain a sparse solution, i.e., only a subset of the arcs is selected among all possible arcs implied by $\bar{R}$. As long as we satisfy the inequality, Y forms an acyclic subgraph. Hence, $\bar{R}$ can be different from adjacency matrix $supp(Y)$ in an optimal solution of (6), and any arc (j, k) such that $\bar{R}_{jk} = 1$ can be selected without creating a cycle. For this reason, we call $\bar{R}$ an *adjacency candidate matrix*. The algorithms proposed later in this section solve (6) by providing different $\bar{\pi}$ and $\bar{R} = adj(\bar{\pi})$ in each iteration. In fact, (6) is separable into m sub problems if F is separable. Let Y_k and Z_k be the k^{th} columns of Y and R, respectively, for node k. Then, solving

$$\min_{Y_k} \ F_k(Y_k) \quad s.t. \quad \bar{R}_k \geq supp(Y_k), \tag{7}$$

for all $k \in J$ gives the same solution as solving (6) if F is separable as $F(Y) = \sum_{k \in J} F_k(Y_k)$.

In Section 3.1, a local improvement algorithm for a given topological order is presented. The algorithm swaps pairs of nodes in the order. In both of the iterative algorithms proposed in Sections 3.2 and 3.3, we use the local improvement algorithm presented in the following section.

3.1 Topological Order Swapping Algorithm

Algorithm 1 tries to improve the solution by swapping the topological order. In each iteration, the algorithm determines the nodes to swap that have order s_1 and s_2 in Line 3, where $s_2 = s_1 + 1$ implies that we select two nodes which are neighbors in the current topological order. Then in Line 4, the actual node indices k_1 and k_2 such that $\pi_{k_1} = s_1$ and $\pi_{k_2} = s_2$ are detected. The condition in Line 5 is to avoid meaningless computation when Y^* is sparse. If $|Y^*_{k_2 k_1}| > 0$, we know for sure that $Y^*_{k_2 k_1} = 0$ after swapping the orders of

k_1 and k_2 and thus we will get a different solution. However, if $Y^*_{k_2k_1} = 0$, we will still have $Y^*_{k_2k_1} = 0$ after the swap forced by the new order. In Line 6, we create a new topological order $\bar{\pi}$ by swapping nodes k_1 and k_2 in π^*. After obtaining adjacency candidate matrix $\bar{R}$ in Line 7, we solve (6) with $\bar{R}$. It is worth noting that, if F is separable, we only need to solve (7) with $k = k_1$ and k_2 because the values of $\bar{R}$ are the same with R^* except for k_1 and k_2 as the order difference was 1 in π^*. In Line 9, we update the best solution if the new solution is better. The iterations continue until there is no improvement in the past m iterations, which implies that we would swap the same nodes if we proceed after this iteration. Algorithm 1 is illustrated by the following toy example.

Algorithm 1 TOSA (Topological Order Swapping Algorithm)

Require: Y', R', π'
Ensure: Best solution Y^*, R^*, π^*
1: $(Y^*, R^*, \pi^*) \leftarrow (Y', R', \pi')$, $t \leftarrow 0$
2: **While** there is an improvement in the past m iterations
3: $t \leftarrow t + 1$, $s_1 \leftarrow (t \bmod (m-1)) + 1$, $s_2 \leftarrow s_1 + 1$
4: $(k_1, k_2) \leftarrow$ node indices satisfying $\pi_{k_1} = s_1$ and $\pi_{k_2} = s_2$
5: **If** $|Y^*_{k_2k_1}| > 0$ **then**
6: $\bar{\pi} \leftarrow \pi^*$, $\bar{\pi}_{k_1} = s_2$, $\bar{\pi}_{k_2} = s_1$
7: $\bar{R} \leftarrow adj(\bar{\pi})$
8: $\bar{Y} \leftarrow$ solve (6) with $\bar{R}$
9: **If** $F(\bar{Y}) < F(Y^*)$ **then** update (Y^*, R^*, π^*)
10: **End if**
11: **End While**

Example 1 *Consider a graph with $m = 4$ nodes. Let us assume that inputs are $\pi' = (2, 3, 1, 4)$ with corresponding order $3 - 1 - 2 - 4$,*

$$Y' = \begin{bmatrix} 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \\ 0.4 & 0.8 & 0.1 & 0 \end{bmatrix}, \text{ and } R' = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

In iteration 1, $t = 1$, $s_1 = 1$, $s_2 = 2$, $k_1 = 3$, and $k_2 = 1$. Hence, we are swapping nodes 3 and 1. Since $|Y^*_{13}| = 0.5 > 0$, $\bar{\pi} = (1, 3, 2, 4)$ is created in Line 6, where the associated order is $1 - 3 - 2 - 4$. If $\bar{\pi}$ gives an improved objective function value, then π^* is updated in Line 9. Let us assume that π^* is not updated. In iteration 2, $t = 2$, $s_1 = 2$, $s_2 = 3$, $k_1 = 1$, and $k_2 = 2$. Since $|Y^*_{21}| = 0$, Lines 6 - 9 are not executed. □

3.2 Iterative Reorering Algorithm

We propose an iterative reordering algorithm based on Property 1, which solves (6) in each iteration aiming to optimize (1). In each iteration of the algorithm, all nodes are sorted based on scores defined by (i) merit scores of the arcs, (ii) historical choice of the arcs (used as weights), (iii) and some random components. Then the sorted node order is directly

used as a topological order. The selected arcs by the topological order give updates on arc weights. Let us first define notation.

ν = uniform random variable on $[\nu_{lb}, \nu_{ub}]$, $\nu_{lb} < 1 < \nu_{ub}$
ρ_{jk} = pre-determined merit score of arc (j,k) for $j \in J, k \in J$
w_{jk} = weight of arc (j,k) for $j \in J, k \in J$
c_k = score of node k, $k \in J$

The range $[\nu_{lb}, \nu_{ub}]$ of the uniform random variable ν balances the randomness and structured scores. Note that ρ_{jk} should be determined based on the data and the characteristic of the problem considered, where larger ρ_{jk} implies that arc (j,k) is attractive. Based on the arc merit scores ρ, the score for node k, $k \in J$, is defined as

$$c_k = \nu \cdot \big(\sum_{j \in J^k} w_{jk}\rho_{jk} \big), \quad k \in J, \tag{8}$$

which can be interpreted as a weighted summation of ρ_{jk}'s multiplied by perturbation random number ν. Hence, nodes with high scores are attractive. Initially, all arcs have equal weights and the weights are updated in each iteration based on the topological order in the iteration. If $\bar{R} = adj(\bar{\pi})$ is the adjacency candidate matrix in the iteration, then, the weights are updated by

$$w_{jk} = w_{jk} + 1, \quad \text{if } \bar{R}_{jk} = 1. \tag{9}$$

The overall algorithmic framework is summarized in Algorithm 2. In Line 1, weights w_{jk}'s are initialized to 1 and $\bar{t}$, which counts the number of iterations without a best solution update, is initialized. Also, a random order π^* of the nodes is generated, and the corresponding solution becomes the best solution. In each iteration, first node scores c_k's are calculated (Line 3), then topological order $\bar{\pi}$ is obtained by sorting the nodes, and finally adjacency candidate matrix $\bar{R}$ is generated (Line 4). Then, in Lines 5 and 6, solution $\bar{Y}$ is obtained by solving (6) with $\bar{R}$ and the best solution is updated if available. In Lines 7 - 10, *TOSA* is executed if the current solution is within a certain percentage α from the best solution. Lines 11 and 12 update $\bar{t}$, and Line 13 updates w_{jk}'s. This ends the iteration and the algorithm continues until $\bar{\pi}$ is converged or there is no update of the best solution in the last t^* iterations. Algorithm 2 is illustrated by the following toy example.

Example 2 *Consider a graph with $m = 3$ nodes. In the current iteration, let us assume that we are given*

$$\rho = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.2 & 0 & 0.2 \\ 0.3 & 0.3 & 0 \end{bmatrix} \text{ and } w = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}.$$

Note that we have $\sum_{j \in J^1} w_{j1}\rho_{j1} = 0.2 \cdot 1 + 0.3 \cdot 2 = 0.8$, $\sum_{j \in J^2} w_{j2}\rho_{j2} = 0.5 \cdot 1 + 0.3 \cdot 1 = 0.8$, and $\sum_{j \in J^3} w_{j3}\rho_{j3} = 0.4 \cdot 2 + 0.2 \cdot 1 = 1$. If random numbers ($\nu$) are 0.9, 1.1, 0.8 for nodes 1,2, and 3, respectively, then by (8)*, $c_1 = 0.9 \cdot 0.8 = 0.72$, $c_2 = 1.1 \cdot 0.8 = 0.88$, and $c_3 = 0.8 \cdot 1 = 0.8$. Then in Line 4, we obtain $\bar{\pi} = (3,1,2)$, with corresponding order $2-3-1$, and $\bar{R} = [0,1,1;0,0,0;0,1,0]$. After obtaining $\bar{Y}$ and updating the best solution in Lines 5-12, the weights are updated by* (9) *as follows.*

Algorithm 2 IR (Iterative Reordering)

Require: Merit score $\rho \in \mathbb{R}^{m\times m}$, termination parameter t^*, *TOSA* execution parameter α
Ensure: Best solution Y^*, R^*, π^*
1: $w_{jk} \leftarrow 1$, $\pi^* \leftarrow$ a random order, $R^* \leftarrow adj(\pi^*)$, $\bar{\pi} \leftarrow \pi^*$, $Y^* \leftarrow$ solve (6) with R^*, $\bar{t} \leftarrow 0$
2: **While** (i) $\bar{\pi}$ is not convergent or (ii) $\bar{t} < t^*$
3: Calculate score c_k by (8)
4: $\bar{\pi} \leftarrow$ sort nodes with respect to c_k, $\bar{R} \leftarrow adj(\bar{\pi})$
5: $\bar{Y} \leftarrow$ solve (6) with $\bar{R}$
6: **If** $F(\bar{Y}) < F(Y^*)$ **then** update (Y^*, R^*, π^*)
7: **If** $F(\bar{Y}) < F(Y^*) \cdot (1+\alpha)$
8: $(Y', R', \pi') \leftarrow TOSA(\bar{Y}, \bar{R}, \bar{\pi})$,
9: **If** $F(Y') < F(Y^*)$ **then** update (Y^*, R^*, π^*)
10: **End If**
11: **If** (Y^*, R^*, π^*) is updated **then** $\bar{t} \leftarrow 0$
12: **Else** $\bar{t} \leftarrow \bar{t} + 1$
13: Update weights by (9)
14: **End While**

$$w_{new} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} + \bar{R} = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 1 \\ 2 & 2 & 0 \end{bmatrix}$$

This ends the current iteration. □

3.3 Gradient Descent Algorithm

In this section, we propose a gradient descent algorithm based on Property 1. The algorithm iteratively executes: (i) moving toward an improving direction by gradients, (ii) DAG structure is recovered and topological order is obtained by a projection step. The algorithm is based on the standard gradient descent framework while the projection step takes care of the acyclicity constraints by generating a topological order from the current (possibly cyclic) solution matrix. In order to distinguish the solutions with and without the acyclicity property, we use the following notation.

$U^t \in \mathbb{R}^{m\times m}$ = decision variable matrix without acyclicity requirement in iteration t
$Y^t \in \mathbb{R}^{m\times m}$ = decision variable matrix satisfying $G(supp(Y^t)) \in \mathcal{A}$ in iteration t

Let γ^t be the step size in iteration t, $\nabla F(Y^t)$ be the derivative of F at Y^t, and $G^t \in \mathbb{R}^{m\times m}$ be a weight matrix that weighs each element. We assume $\|\nabla F(Y^t)\|_\infty \leq M_1$ for a constant M_1, where $\|\cdot\|_\infty$ is the uniform (infinity) norm. The update formula

$$U^t = Y^t - \gamma^t\big[\nabla F(Y^t) \circ G^t\big], \quad t \geq 0, \tag{10}$$

updates Y^t based on the weighted gradient, where $\circ$ represents the entrywise or Hadamard product of the two matrices. Given topological order π^t, we define G^t as

$$G^t_{jk} = \left(1 + \frac{1}{\pi^t_k}\right)^{\pi^t_k}, \quad j \in J^k \tag{11}$$

to balance gradients of the nodes with different orders (small and large values of π^t_k). For nodes k_1 and k_2 with $\pi^t_{k_1} = 1$ and $\pi^t_{k_2} = m$, most of the gradients for node k_1 are zero and most of the gradients for node k_2 are nonzero. Weight (11) tries to adjust this gap. Note that we have $2 \leq G^t_{jk} \leq e$ for any large m. Since U^t may not satisfy acyclic constraints, in order to obtain a DAG, the algorithm needs to solve the projection problem

$$Y^* = \text{argmin}_Y \ \|Y - U^t\|_2^2 \quad s.t. \quad G(supp(Y)) \in \mathcal{A}, \tag{12}$$

where $\|\cdot\|_2$ is the L_2 norm.

Proposition 2 *If U^t is arbitrary, then optimization problem* (12) *is $\mathcal{NP}$-hard.*

Proof Recall that feedback arc set is $\mathcal{NP}$-complete (Karp, 1972) and maximum acyclic subgraph is the dual of the feedback arc set problem. With $U^t = 1$, (12) becomes the weighted maximum acyclic subgraph problem. Therefore, (12) is $\mathcal{NP}$-complete. ■

Because solving (12) to optimality does not guarantee an optimal solution for (1), we use a greedy strategy to solve (12). The greedy algorithm, presented in Algorithm 3, sequentially determines and fixes the topological order of a node where in each iteration the problem is solved optimally given the currently fixed nodes and corresponding orders. The detailed derivations of the algorithm and the proof that each iteration is optimal, given already fixed node orders, are available in Appendix 6. In other words, we show that Line 3 is 'locally' optimal, i.e., it selects the best next node given that the order $q+1, q+2, \cdots, m$ is fixed. In each iteration, in Line 3, the algorithm first calculates score $\sum_{j \in \bar{J}} (\bar{U}^t_{jk})^2$ for each node k in $\bar{J}$ and picks node k^* with the minimum value. Then, in Line 4, the order of the selected node is fixed to q. The fixed node is then excluded from the active set $\bar{J}$ and iterate q is decreased by 1 in Line 5. At the end of the algorithm, we can determine $\bar{Y}$ based on the order $\bar{\pi}$ determined and (18) in Appendix 6. We illustrate Algorithm 3 by the following example.

Example 3 *Consider a graph with $m = 3$ nodes. Given U^t, the algorithm returns $\bar{Y}$ presented in the following.*

$$U^t = \begin{bmatrix} 0 & 1 & 2 \\ 4 & 0 & 2 \\ 5 & 2 & 0 \end{bmatrix} \quad \bar{Y} = \begin{bmatrix} 0 & 0 & 0 \\ 4 & 0 & 2 \\ 5 & 0 & 0 \end{bmatrix}$$

Algorithm 3 starts with $q = 3$ and $\bar{J} = \{1, 2, 3\}$. In iteration 1, node 2 is selected to have $\pi_2 = 3$ based on argmin$\{4^2 + 5^2, 1^2 + 2^2, 2^2 + 2^2\}$. Then, set $\bar{J}$ and integer q are updated to $\bar{J} = \{1, 3\}$ and $q = 2$. In iteration 2, node 3 is selected to have $\pi_3 = 2$ based on argmin$\{5^2, 2^2\}$. Then, set $\bar{J}$ and integer q are updated to $\bar{J} = \{1\}$ and $q = 1$. In iteration 3, node 1 is selected. Hence, we have node order 1-3-2 and we obtain $\bar{Y}$ presented above with objective function value $\|\bar{Y} - U^t\|_2^2 = 1^2 + 2^2 + 2^2 = 9$. □

Algorithm 3 Greedy

Require: $U^t \in \mathbb{R}^{m\times m}$
Ensure: $\bar{Y}$ feasible to (12), topological order $\bar{\pi}$
1: $q \leftarrow m$, $\bar{J} \leftarrow J$
2: **While** $\bar{J} \neq \emptyset$
3: $\quad k^* = \text{argmin}_{k\in\bar{J}}\Big\{\sum_{j\in\bar{J}}(U^t_{jk})^2\Big\}$
4: $\quad \bar{\pi}_{k^*} = q$
5: $\quad \bar{J} \leftarrow \bar{J}\setminus\{k^*\}, q \leftarrow q-1$
6: **End While**
7: Determine $\bar{Y}$ by (18) in Appendix 6

The overall gradient descent algorithm for (1) is presented in Algorithm 4. In Line 1, the algorithm generates a random order π^* and obtain corresponding R^* and Y^* and save them as the best solution. In each iteration of the loop, Lines 3-6 follow the standard gradient descent algorithm. The weighted gradient H^t is calculated in Line 3, and the step size is determined in Line 4 based on the ratio between $\max_{j\in J^k,k\in J}|H^t_{jk}|$ and $\max_{j\in J^k,k\in J}|Y^t_{jk}|$. In Line 5, the solution is updated based on the weighted gradient and, in Line 6, the greedy algorithm is used to obtain the projected solution and the topological order. Observe that we do not directly use the projected solution. This is because the projected solution is not necessarily optimal given π^{t+1}. Hence, in Line 7, a new solution Y^{t+1} is obtained based on π^{t+1}. In Lines 9 - 12, $TOSA$ is executed if the current solution is within a certain percentage from the current best solution. Lines 13 and 14 update $\bar{t}$ and Line 15 copies Y^* to Y^{t+1} if $\bar{t} \geq t_2^*$ in order to focus on the solution space near Y^*. The algorithm continues until Y^t is convergent or $\bar{t} \geq t_1^*$.

In gradient based algorithms, it is common to have γ^t depend only on t, but in our case dependency on H^t and Y^t is justifiable since we multiply the gradient by G^t. We next show the convergence of Y^t in Algorithm 4 when $t_1^* = t_2^* = \infty$. This makes the algorithm not to terminate unless Y^t has converged and modification of Y^t in Line 15 is not executed. Further, we assume the following for the analysis.

Assumption 1 *For any non-zero element Y^t_{jk}, $j,k \in J$, of Y^t in any iteration t, we assume $\varepsilon < |Y^t_{jk}| < M_2$, where ε is a small positive number and M_2 is a large enough number.*

Note that Assumption 1 is a mild assumption, as ignoring near-zero values of Y^t happens in practice anyway due to finite precision. For notational convenience, let $L^t = \gamma^t \nabla F(Y^t) \circ G^t = \gamma^t H^t$ be the second term in (10). Then, U^t can be written as $U^t = Y^t - L^t = Y^t - \gamma^t H^t$. In the following lemma, we show that the node orders converge.

Lemma 3 *If t is sufficiently large satisfying $\sqrt{t} > \frac{(M_1 e)^2}{\varepsilon(\sqrt{M_2^2+\varepsilon^2/m}-M_2)}$, then $\pi^t = \pi^{t+1}$.*

Proof Recall that Y^t is obtained by solving (20) and we know the corresponding node order π^t. Let $k_1, k_2, \cdots, k_m$ be the node indices defined based on π^t. In other words, node k_1 appears first, followed by nodes k_2, k_3, and so on in the topological order π^t. In the proof, we

Algorithm 4 GD (Gradient Descent)

Require: Parameters t_1^* and t_2^*, *TOSA* execution parameter α
Ensure: Best solution Y^*, R^*, π^*
$t \leftarrow 1$, $\bar{t} \leftarrow 0$, $\pi^* \leftarrow$ a random order, $R^* \leftarrow adj(\pi^*)$, $Y^* \leftarrow$ solve (6) with R^*
While (i) Y^t is not convergent or (ii) $\bar{t} < t_1^*$
  $H^t \leftarrow \nabla F(Y^t) \circ G^t$, G^t defined in (11)
  $\gamma^t \leftarrow \frac{\|H^t\|_\infty}{\|Y^t\|_\infty} / \sqrt{t}$
  $U^t \leftarrow Y^t - \gamma^t H^t$
  $\pi^{t+1} \leftarrow$ *Greedy*(U^t)
  $Y^{t+1} \leftarrow$ solve (6) with $R^{t+1} = adj(\pi^{t+1})$
  If $F(Y^{t+1}) < F(Y^*)$ **then** $(Y^*, R^*, \pi^*) \leftarrow (Y^{t+1}, R^{t+1}, \pi^{t+1})$
  If $F(\bar{Y}) < F(Y^*) \cdot (1+\alpha)$
    $(Y', R', \pi') \leftarrow TOSA(Y^{t+1}, R^{t+1}, \pi^t)$,
    If $F(Y') < F(Y^*)$ **then** $(Y^*, R^*, \pi^*) \leftarrow (Y', R', \pi'), (Y^{t+1}, R^{t+1}, \pi^{t+1}) \leftarrow (Y', R', \pi')$
  End If
  If (Y^*, R^*, π^*) is updated **then** $\bar{t} \leftarrow 0$
  Else $\bar{t} \leftarrow \bar{t} + 1$
  If $\bar{t} \geq t_2^*$ **then** $Y^{t+1} \leftarrow Y^*$
  $t \leftarrow t + 1$
End While

show that there is no change in the node order when the condition $\sqrt{t} > \frac{(M_1 e)^2}{\varepsilon(\sqrt{M_2^2 + \varepsilon^2/m} - M_2)}$ is met, where M_1 and M_2 are the upper bounds for $\|\nabla F(Y^t)\|_\infty$ and $\|Y^t\|_\infty$, respectively, as assumed. We first derive

$$\|L^t\|_\infty = \left\|\gamma^t H^t\right\|_\infty = \left\|\frac{1}{\sqrt{t}} \frac{\|H^t\|_\infty}{\|Y^t\|_\infty} H^t\right\|_\infty \leq \frac{1}{\sqrt{t}} \frac{\left(\|H^t\|_\infty\right)^2}{\|Y^t\|_\infty} < \frac{1}{\sqrt{t}} \frac{(M_1 e)^2}{\varepsilon}, \tag{13}$$

where the last inequality holds since (i) $\|Y^t\|_\infty > \varepsilon$ by Assumption 1, (ii) $\|H^t\|_\infty = \|\nabla F(Y^t) \circ G^t\|_\infty \leq M_1 e$ because $\|\nabla F(Y^t)\|_\infty \leq M_1$ by the assumption and $\|G^t\|_\infty \leq e$, where e is natural number.

Now let us consider $q = \bar{q}$ in Algorithm 3 to decide node order $\bar{q}$ in iteration $t+1$ and assume $\pi_{k_r}^t = \pi_{k_r}^{t+1}$ for $r = m, m-1, \cdots, \bar{q}-1$. Note that we currently have $\bar{J} = \{k_1, k_2, \cdots, k_{\bar{q}}\}$.

1. For $k_{\bar{q}}$, we derive $\sum_{j \in \bar{J}} (U_{jk_{\bar{q}}}^t)^2 = \sum_{j \in \bar{J}} (Y_{jk_{\bar{q}}}^t - L_{jk_{\bar{q}}}^t)^2 = \sum_{j \in \bar{J}} (L_{jk_{\bar{q}}}^t)^2 < m\left[\frac{(M_1 e)^2}{\sqrt{t}\varepsilon}\right]^2$, where the second equality holds since $Y_{jk_{\bar{q}}}^t = 0$ for all $j \in \bar{J}$ since $\pi_{k_{\bar{q}}} = \bar{q}$ and no arc can be used to the nodes in $\bar{J}$, and the last inequality holds due to (13) and $|\bar{J}| \leq m$.

2. For all other nodes $k_r \in \bar{J} \setminus \{k_{\bar{q}}\}$, we derive

$$\begin{array}{rcl}
\sum_{j\in\bar{J}}(U^t_{jk_r})^2 &=& \sum_{j\in\bar{J}}(Y^t_{jk_r}-L^t_{jk_r})^2 \\
&=& \sum_{j\in\bar{J}}(Y^t_{jk_r})^2+\sum_{j\in\bar{J}}(L^t_{jk_r})^2-2\sum_{j\in\bar{J}}Y^t_{jk_r}\cdot L^t_{jk_r} \\
&>& \sum_{j\in\bar{J}}(Y^t_{jk_r})^2-2\sum_{j\in\bar{J}}Y^t_{jk_r}\cdot L^t_{jk_r} \\
&>& \varepsilon^2-2\sum_{j\in\bar{J}}Y^t_{jk_r}\cdot L^t_{jk_r} \\
&\geq& \varepsilon^2-2\sum_{j\in\bar{J}}|Y^t_{jk_m}\cdot L^t_{jk_r}| \\
&>& \varepsilon^2-2M_2\frac{m(M_1e)^2}{\sqrt{t}\varepsilon}
\end{array}$$

where the fourth line holds due to $|Y^t_{jk_r}|>\varepsilon$ by Assumption 1, and the sixth line holds due to $|Y^t_{jk}|\leq M_2$, $|\bar{J}|\leq m$, and $|L^t_{jk_r}|<\frac{(M_1e)^2}{\sqrt{t}\varepsilon}$ by (13).

Combining the two results for $k_{\bar{q}}$ and $k_r\in\bar{J}\setminus\{k_{\bar{q}}\}$, we obtain $\sum_{j\in\bar{J}}(U^t_{jk_{\bar{q}}})^2<m\big[\frac{(M_1e)^2}{\sqrt{t}\varepsilon}\big]^2<\varepsilon^2-2M_2\frac{m(M_1e)^2}{\sqrt{t}\varepsilon}<\sum_{j\in\bar{J}}(U^t_{jk_r})^2$, for any $r\in\{1,2,\cdots,\bar{q}-1\}$, where the second inequality holds due to the condition $\sqrt{t}>\frac{(M_1e)^2}{\varepsilon(\sqrt{M_2^2+\varepsilon^2/m}-M_2)}$. The result implies that we must have $\pi^{t+1}_{k_{\bar{q}}}=\pi^t_{k_{\bar{q}}}=\bar{q}$ by Line 3 in Algorithm 3.

Note that, when $\bar{q}=m$, we have $J=\bar{J}$ and the assumption of $\pi^t_{k_r}=\pi^{t+1}_{k_r}$ for $r=m,m-1,\cdots,\bar{q}-1$ automatically holds. By iteratively applying the above derivation technique from $q=m$ to 1, we can show that $\pi^t=\pi^{t+1}$. ■

When (6) is solved with the identical node orders, the resulting solutions are equivalent. Hence, the following proposition holds.

Proposition 4 *In Algorithm 4, Y^t converges in t.*

4. Estimation of Gaussian Bayesian Networks

In this section, we introduce the Gaussian Bayesian network learning problem, which follows the form of (1). The goal is to learn or estimate unknown structure between the nodes of a graph, where the error is normally distributed. The network can be estimated by optimizing a score function, testing conditional independence, or a mix of the two, as described in Section 1. Among the three categories, we select the score based approach with the L_1-penalized least square function recently studied in Han et al. (2016).

Let $X\in\mathbb{R}^{n\times m}$ be a data set with n observations and m features. Let $I=\{1,\cdots,n\}$ and $J=\{1,\cdots,m\}$ be the index set of observations and features, respectively. For each $k\in J$, we build a regression model in order to explain feature k using a subset of variables in J^k. In other words, we set feature k as the response variable and a sparse subset of J^k as explanatory variables of the regression model for variable k. In order to obtain a subset of J^k, the LASSO penalty function is added. Considering regression models for all $k\in J$ together, the problem can be represented on a graph. Each feature is a node in the graph, and the directed arc from node j to node k represents explanatory and response variable relationship between node j and k. The goal is to minimize the sum of penalized SSE over all regression models for $k\in J$, while the selected arcs do not create a cycle.

Let β_{jk}, $j \in J^k$, $k \in J$, be the coefficient of attribute j for dependent variable k. Then the problem can be written as

$$\min_{\beta} \frac{1}{n} \sum_{k \in J} \sum_{i \in I} (x_{ik} - \sum_{j \in J^k} \beta_{jk} x_{ij})^2 + \lambda \sum_{k \in J} \sum_{j \in J^k} |\beta_{jk}| \quad s.t. \quad G(supp(\beta)) \in \mathcal{A}, \tag{14}$$

which follows the form of (1). In Han et al. (2016), individual weights are used for the penalty term, i.e., $\lambda \sum_{k \in J} \sum_{j \in J^k} w_{jk} |\beta_{jk}|$, however, in the computational experiment, we set all weights equal to 1 for simplicity.

Let $Z_{jk} = 1$ if attribute j is used for dependent variable k and $Z_{jk} = 0$ otherwise. Then, we can formulate MIPto for (14) as

$$\begin{aligned}
\min \quad & \frac{1}{n} \sum_{k \in J} \sum_{i \in I} (x_{ik} - \sum_{j \in J^k} \beta_{jk} x_{ij})^2 + \lambda \sum_{k \in J} \sum_{j \in J^k} |\beta_{jk}| && && \text{(15a)} \\
s.t. \quad & |\beta_{jk}| \leq M Z_{jk}, && j \in J^k, k \in J, && \text{(15b)} \\
& Z_{jk} - m Z_{kj} \leq \sum_{r \in J} r(O_{kr} - O_{jr}), && j \in J^k, k \in J, && \text{(15c)} \\
& Z_{jk} + Z_{kj} \leq 1, && j \in J^k, k \in J, && \text{(15d)} \\
& \sum_{q \in J} O_{kq} = 1, && k \in J, && \text{(15e)} \\
& \sum_{k \in J} O_{kq} = 1, && q \in J, && \text{(15f)} \\
& Z, O \in \{0,1\}^{m \times m}, && && \text{(15g)} \\
& \beta_{jk} \text{ not restricted}, && j \in J^k \cup \{0\}, k \in J, && \text{(15h)}
\end{aligned}$$

where M is a large constant. Note that (15b) is the linear constraint corresponding to $Z = supp(Y)$ in (5). Similarly, (15a), (15b), (4c) - (4e) and (15h) can be used to formulate MIPin for (14). For MIPcp, (15a), (15b), (15h), and the constraints in (2) can be used for (14).

Note that M in (15b) plays an important role in computational efficiency and optimality. If M is too small, the MIP model cannot guarantee optimality. If M is too large, the solution time can be as large as enumeration. The algorithm for getting a valid value for M in Park and Klabjan (2013) can be used. However, the valid value of big M for multiple linear regression is often too large (Park and Klabjan, 2013). For (15), we observed that a simple heuristic presented in Section 5 works well.

In each iteration of IR (Algorithm 2) and GD (Algorithm 4), we are given topological order $\bar{\pi}$ and matrix $\bar{R} = adj(\bar{\pi})$. Let $S^k = \{j \in J^k | \bar{R}_{jk} = 1\}$ be the set of selected candidate arcs for dependent variable k. Given fixed $\bar{R}$, (14) is separable into m LASSO linear regression problems

$$\min \quad \frac{1}{n} \sum_{i \in I} (x_{ik} - \sum_{j \in S^k} \beta_{jk} x_{ij})^2 + \lambda \sum_{j \in S^k} |\beta_{jk}|, \quad k \in J. \tag{16}$$

5. Computational Experiment

For all computational experiments, a server with two Xeon 2.70GHz CPUs and 24GB RAM is used. Although there are many papers studying Bayesian network learning with various error measures and penalties, here we focus on minimizing the LASSO type objective (SSE and penalty) and we picked one of the latest paper of Han et al. (2016) with the same objective function as a benchmark.

The MIP models MIPcp, MIPin, and MIPto are implemented with CPLEX 12.6 in C#. For MIPcp, instead of implementing the original cutting plane algorithm, we use CPLEX Lazy Callback, which is similar to the cutting plane algorithm. Instead of solving (3) to optimally from scratch in each iteration, we solve (2) with Lazy Callback, which allows updating (adding) constraints (cycle prevention constraints) in the process of the branch and bound algorithm whenever an integer solution with cycles is found. Given a solution with the cycles, we detect all cycles and add cycle prevention constraints for the detected cycles.

For MIPcp, MIPin, and MIPto, we set big M as follows. Given λ, we solve (14) without acyclic constraints. Hence, we are allowed to use all arcs in J^k for each model $k \in J$. Then, we obtain the estimated upper bound for big M by

$$M = 2\Big(\max_{j \in J^k, k \in J} |\beta_{jk}|\Big). \tag{17}$$

We observed that the above formula gives large enough big M for all cases in the following experiment. In Appendix 6, we present comparison of regression coefficients of implanted network (DAG) with big M values by (17). The result shows that the big M value in (17) is always valid for all cases considered.

We compare our algorithms and models with the algorithm in Han et al. (2016), which we denote as DIST here. Their algorithm starts with neighborhood selection (NS), which filters unattractive arcs and removes them from consideration. The procedure is specifically developed for high dimensional variable selection when m is much larger than n. In our experiment, many instances considered are not high dimensional and some have dense solutions. Further, by filtering arcs, there exists a probability that an arc in the optimal solution can be removed. Hence, we deactivated the neighborhood selection step of their original algorithm, where the R script of the original algorithm is available on the journal website.

For GD and IR, the algorithms are written in R (R Core Team, 2016). We use glmnet package (Friedman et al., 2010) function glmnet for solving LASSO linear regression problems in (16). For IR, we use parameters $\alpha = 0.01$, $t^* = 10$, $\nu_{lb} = 0.8$, and $\nu_{ub} = 1.2$. For GD, we use parameters $\alpha = 0.01$, $t_1^* = 10$, and $t_2^* = 5$. Because both GD and IR start with a random solution, they perform different with different random solutions. Further, since we observe that the execution time of GD and IR are much faster than DIST, we decided to run GD and IR with 10 different random seeds and report the best solution. To emphasize the number of different random seeds for GD and IR, we use the notation GD10 and IR10 in the rest of the section.

We first test all algorithms with synthetic instances generated using R package pcalg (Kalisch et al., 2012). Function randomDAG is used to generate a DAG and function rmvDAG is used to generate multivariate data with the standard normal error distribution. First,

a DAG is generated by randomDAG function. Next, the generated DAG and random coefficients are used to create each column (with standard normal error added) by rmvDAG function which uses linear regression as the underlying model. After obtaining the data matrix from the package, we standardize each column to have zero mean with standard deviation equal to one. The DAG used to generate the multivariate data is considered as the true structure or true arc set while it may not be the optimal solution for the score function. The random instances are generated for various parameters described in the following.

m: number of features (nodes)
n: number of observations
s: expected number of true arcs per node
d: expected density of the adjacency matrix of the true arcs

By changing the ranges of the above parameters, three classes of random instances are generated.

Sparse data sets: The expected total number of true arcs is controlled by s and most of the instances have a sparse true arc set. We use $n \in \{100, 200, 300\}$, $m \in \{20, 30, 40, 50\}$, and $s \in \{1, 2, 3\}$ to generate 10 instances for each (n, m, s) triplet. This yields a total of 360 random instances.

Dense data sets: The expected total number of true arcs is controlled by d and most of the instances have a dense true arc set compared with the sparse data sets. We use $n \in \{100, 200, 300\}$, $m \in \{20, 30, 40, 50\}$, and $d \in \{0.1, 0.2, 0.3\}$ to generate 10 instances for each (n, m, d) triplet, and thus we have a total of 360 random instances.

High dimensional data sets: The instances are high dimensional ($m \geq n$) and very sparse. The expected total number of true arcs is controlled by s. We use $n = 100$ and $m \in \{100, 150, 200\}$, and $s \in \{0.5, 1, 1.5\}$ to generate 10 instances for each (m, s) pair, which yields a total of 90 random instances.

We use four λ values differently defined for each data set in order to cover the expected number of arcs with the four λ values. For each sparse instance, we solve (14) with $\lambda \in \{1, 0.5, 0.1, 0.05\}$. For dense data sets, a wide range of λ values are needed to obtain selected arc sets that have similar cardinalities with the true arc sets. Hence, for each dense instance, instead of fixed values over all instances in the set, we use λ values based on expected density d: $\lambda = \lambda_0 \cdot 10^{-(10 \cdot d - 1)}$, where $\lambda_0 \in \{1, 0.1, 0.01, 0.001\}$. For each high dimensional instance, we use $\lambda \in \{1, 0.8, 0.6, 0.4\}$. Observe that the expected densities of the adjacency matrices vary across the three data sets. The sparse instances have expected densities between 0.02 and 0.15, the dense instances have expected densities between 0.1 and 0.3, and the high dimensional instances have expected densities between 0.002 and 0.015. Hence, different ranges of λ values are necessary.

For all of the results presented in this section, we present the average performance by n, m, s, d and λ. For example, the result for $n = 100$ and $m = 20$ are the averages of 120 and 90 instances, respectively. In all of the comparisons, we use the following metrics.

time: computation time in seconds
δ_{sol}: relative gap (%) from the best objective value among the compared algorithms or models. For example, if we compare the three MIP models, δ_{sol} of an MIP model is

the relative gap from the best of the three objective function values obtained by the MIP models.
$\|z\|_0$: number of arcs selected (number of nonzero regression coefficients β_{jk}'s)

In comparing the performance metrics, we use plot matrices. In each Figure 1, 2, 3, 5, and 6, multiple bar plots form a matrix. The rows of the plot matrix correspond to performance metrics and the columns stand for parameters used for result aggregation. For example, the left top plot in Figure 1 shows execution times of the algorithms where the results are aggregated by n (the number of observations), because the first row and first column of the plot matrix in Figure 1 are for execution times and n, respectively.

In Section 5.1, we compare the performance of iterative algorithms GD10 and IR10 and the benchmark algorithm DIST. In Section 5.2, we compare the performance of MIP models MIPcp, MIPin, and MIPto. We also compare all models and algorithms with a subset of the synthetic instances in Section 5.2. Finally, in Section 5.4, we solve a popular real instance of Sachs et al. (2005) in the literature.

5.1 Comparison of Iterative Algorithms

In this section, we compare the performance of GD10, IR10, and DIST by *time*, δ_{sol}, and $\|z\|_0$ for each of the three data sets.

In Figure 1, the result for the sparse data sets is presented. The bar plot matrix presents the performance measures aggregated by n, m, s, and λ.

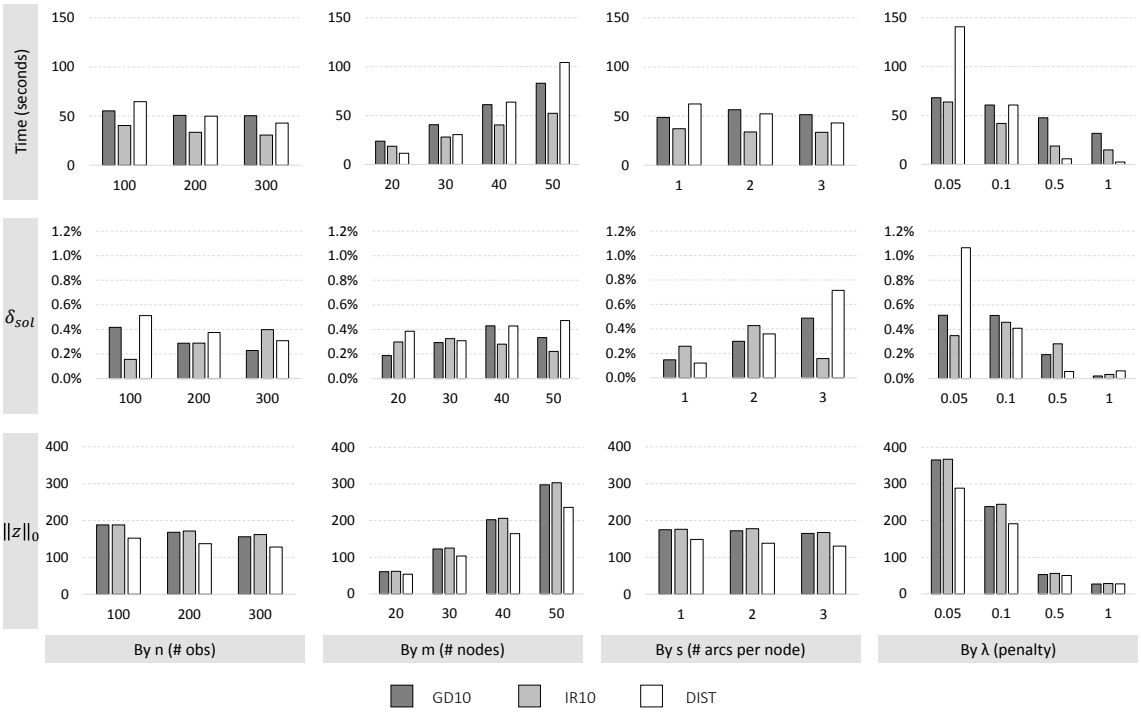

Figure 1: Performance of GD10, IR10, and DIST (sparse data)

The computation time of all three algorithms increases in increasing m and decreasing λ, where the computation time of DIST increases faster than the other two. The computation time of DIST is approximately 10 times faster than the GD10 time when $\lambda = 1$, but 2 times slower when $\lambda = 0.05$. With increasing n, the computation times of all algorithms stay the same or decrease. This can be seen counter-intuitive because larger instances do not increase time. However, a larger number of observations can make predictions more accurate and could reduce search time for unattractive subsets. Especially, the computation time of DIST decreases in increasing n. We think this is because more observations give better local selection in the algorithm when adding and removing arcs. The number of selected arcs ($\|z\|_0$) of GD10 and IR10 is greater than DIST for all cases because the topological order based algorithms are capable of using the maximum number of arcs $\left(\frac{m(m-1)}{2}\right)$, while arc selection based algorithms, such as DIST, are struggling to select many arcs without violating acyclic constraints. In terms of the solution quality, all algorithms have δ_{sol} less than 1.2% and perform good. However, we observe several trends. As λ decreases (required to select more arcs), GD10 and IR10 start to outperform. We also observe that, as the problem requires to select more arcs (increasing m, increasing s, and decreasing λ), GD10 and IR10 perform better. As n increases, δ_{sol} of GD10 and DIST decrease, whereas δ_{sol} of IR10 increases.

The result for the dense data sets is presented in Figure 2. The bar plot matrix presents the performance measures aggregated by n, m, d, and λ_0. Recall that, for the dense data set, we solve (14) with $\lambda = \lambda_0 \cdot 10^{-(10 \cdot d - 1)}$ and $\lambda_0 \in \{1, 0.1, 0.01, 0.001\}$. For simplicity of presenting the aggregated result, we use λ_0 in the plot matrix, while λ values are used for actual computation.

The computation time of all three algorithms again increases in increasing m and decreasing λ, where the computation time of DIST increases faster than the other two. Compare to the result for the sparse data sets, the execution times are all larger for the dense data set. The number of selected arcs ($\|z\|_0$) of GD10 and IR10 is again greater than DIST for all cases, where $\|z\|_0$ is twice larger for GD10 and IR10 when m or d is large, or λ is small. In terms of the solution quality, GD10 and IR10 outperform in most of the cases, while δ_{sol} values of DIST increase fast in changing m, d, and λ. The values of δ_{sol} for all algorithms are larger than the sparse data sets result. GD10 and IR10 are better for most of the cases. In general, we again observe that GD10 and IR10 perform better when the problem requires to select more arcs.

The result for the high dimensional data sets is presented in Figure 3. The bar plot matrix presents the performance measures aggregated by m, s, and λ, while n is excluded from the matrix as we fixed n to 100.

The computation time of all three algorithms again increases in increasing m and decreasing λ. However, unlike the previous two sets, the computation times of GD10 and IR10 increase faster than DIST. This is due to the efficiency of the topological order based algorithms. When a very small portion of the arcs should be selected in the solution, topological orders are not informative. For example, consider a graph with three nodes A, B, C and assume that only one arc, (A, B), is selected due to a large penalty. For this case, three topological orders $A-B-C$, $A-C-B$, and $C-A-B$ can represent the selected arc. The third row of Figure 3 shows that $\|z\|_0$ of the three algorithms are very similar, while DIST has much smaller values for the previous two data sets. This implies that arc based search

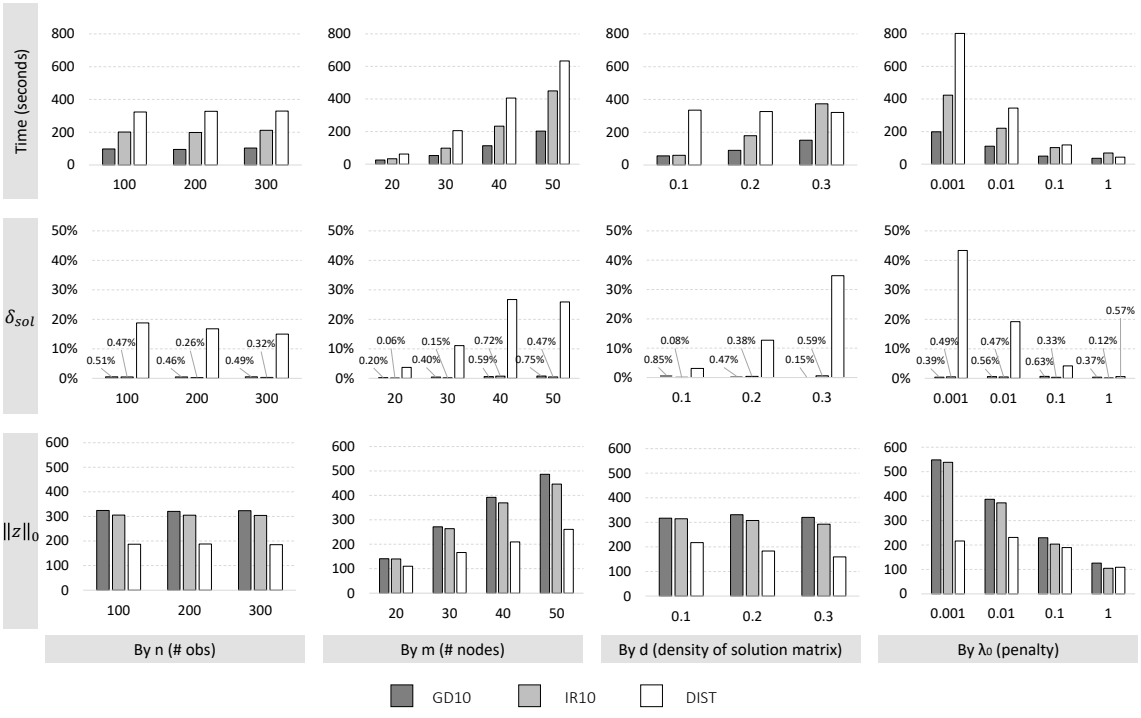

Figure 2: Performance of GD10, IR10, and DIST (dense data)

by DIST does not have difficulties preventing cycles and the algorithm can decide whether to include arcs easier. The comparison of δ_{sol} values also show that arc based search is competitive. Although all algorithms have δ_{sol} values less than 0.5%, we find clear evidence that the performance of IR10 decreases in increasing m and s and decreasing λ. Although the δ_{sol} values of GD10 and DIST are similar, considering the fast computing time of DIST, we recommend to use DIST for very sparse high dimensional data.

In Figure 4, we present combined results of all three data sets by relating solution densities and δ_{sol}. Observe that, for each quadruplet of $n, m, s(\text{or } d), \lambda$, we have results from 10 random instances for each algorithm. Value δ_{sol} is the average of the 10 results for each quadruplet and for each algorithm, $AvgDen$ is the average density of the adjacency matrices of the 10 results and the three algorithms. In Figure 4, we present a scatter plot of $ln(1+AvgDen)$ and $ln(1+100\cdot\delta_{\text{sol}})$. Each point in the plot is the average of 10 results by an algorithm and each algorithm has 324 points displayed [1]. The numbers in the parenthesis along the axes are the corresponding values of δ_{sol} and $AvgDen$. In the plot, we first observe that the algorithms perform similarly when the solutions are sparse and the δ_{sol} values have large variance when the solutions are dense. When the log transformed solution density is less than 2, the average δ_{sol} values of GD10, IR10, and DIST are 0.06%, 0.15%, and 0.04%, respectively. However, the solution quality of DIST drastically decreases as the solutions become denser. This makes sense because sparse solutions can be efficiently searched by

1. $324 = (3\cdot4\cdot3\cdot4)+(3\cdot4\cdot3\cdot4)+(1\cdot3\cdot3\cdot4)$, where the three terms are for the three data sets and each term is obtained by multiplying the number of parameters $n, m, s(d)$, and λ, respectively.

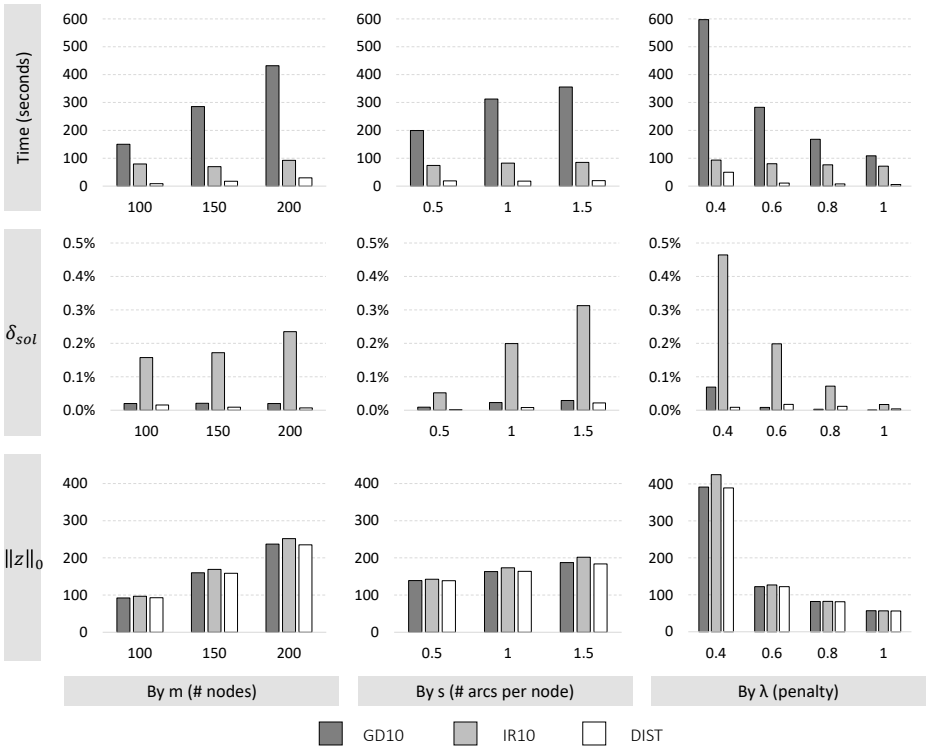

Figure 3: Performance of GD10, IR10, and DIST (high dimensional data)

arc-based search, while dense solutions are not easy to obtain by adding or removing arcs one by one. This also explains the relatively small and large δ_{sol} values for dense and spares solutions, respectively, by the topological order based algorithms. Between GD10 and IR10, we do not observe a big difference.

5.2 Comparison of MIP Models

In this section, we compare the performance of MIPto, MIPin, and MIPcp using *time*, δ_{sol}, and $\|z\|_0$ and the following additional metric.

δ_{IP}: the optimality gap (%) obtained by CPLEX within allowed $15 \cdot m$ seconds

Due to scalability issues of all models, we only use the sparse data with $m = 20, 30, 40$. We also limit $n = 100$. For all instances, we use the $15 \cdot m$ seconds time limit for CPLEX. For example, we have a time limit of 300 seconds for instances with $m = 20$.

The result is presented in Figure 5. Comparing the time of all models with the time limit for CPLEX, we observe that MIPin and MIPcp were able to terminate with optimality for several instances when $m = 20$ and $\lambda = 1$. This implies that MIPin and MIPcp are efficient when the problem is small and the number of selected arcs $\|z\|_0$ is small. However, in general, δ_{IP} values tend to be consistent with different models, while they increase in increasing m and s and in decreasing λ for all three MIP models. The execution times of all models increase in increasing m and s, and in decreasing λ. The same trend can be

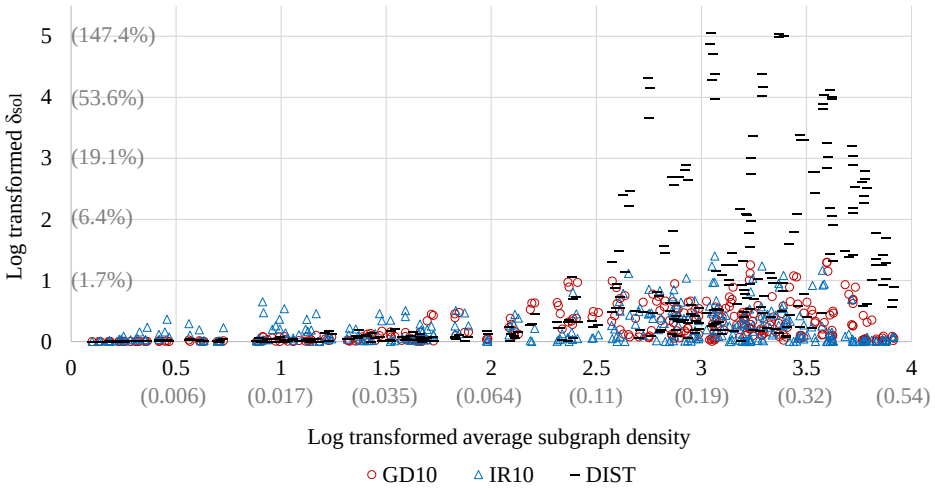

Figure 4: Scatter plot of δ_{sol} and average solution densities

found for δ_{IP} for all models. By comparing δ_{sol} values, we observe that MIPin is best when $m = 20$ and 30. However, the performance of MIPin drops drastically as m and s increase and λ decreases. Actually, MIPin fails to obtain a reasonably good solution within the time limit for several instances. This gives large δ_{sol} values and increases the average. The δ_{sol} values of MIPto are smaller than MIPcp when m and s are small, while MIPcp outperforms when $m = 40$ or $s = 3$.

5.3 Comparison of all MIP Models and Algorithms

In Figure 6, we compare all models and algorithms for selected sparse instances with $n = 100$ and $m \in \{20, 30, 40\}$, which were used to test MIP models. In the plot matrix, we show the average computation time and δ_{sol} (gap from the best objective value among the six models and algorithms) by m, s, and λ. Note that δ_{sol} values of a few MIPin results are not fully displayed in the bar plots due to their large values. Instead, the actual numbers are displayed next to the corresponding bar. The result shows that MIP models spent more time while the solution qualities are inferior in general. The values of δ_{sol} for the MIP models are competitive only when λ is large, which requires sparse solution. However, even for this case, MIP models spend longer time than the algorithms. Hence, ignoring the benefit of knowing and guaranteeing optimality by the MIP models, we conclude that the algorithms perform better for all cases.

The primary reason for the inferior performance of the MIP models is the difficulty of solving integer programming problems. Further, all of the MIP models have at least $O(m^2)$ binary variables and $O(m^2)$ constraints and the problem complexity grows fast. Finally, large values of big M in (15b) make the problem even more difficult. Due to non-tight values of big M, fathoming does not happen frequently in the branch and bound procedure. Hence, at least for sparse Gaussian Bayesian network learning, the MIP models may not be the best option unless other complicated constraints, which cannot be easily dealt with iterative algorithms, are needed.

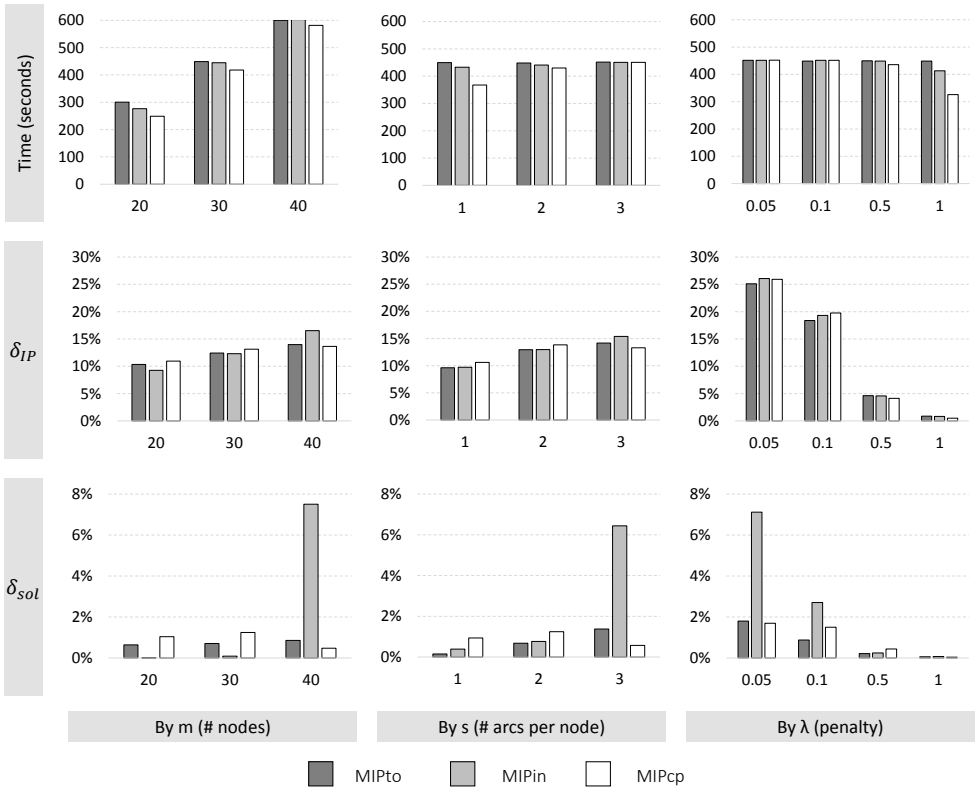

Figure 5: Performance of MIPto, MIPin, and MIPcp (sparse data with $n = 100$ and $m \in \{20, 30, 40\}$)

5.4 Real Data Example

In this section, we study the flow cytometry data set from Sachs et al. (2005) by solving (14). The data set has been studied in many works including Friedman et al. (2008), Shojaie and Michailidis (2010), Fu and Zhou (2013), and Aragam and Zhou (2015). The data set is often used as a benchmark as the casual relationships (underlying DAG) are known. It has $n = 7466$ cells obtained from multiple experiments with $m = 11$ measurements. The known structure contains 20 arcs. For the experiment, we standardize each column to have zero mean with standard deviation equal to one.

In Table 2, we compare the performance of the three algorithms GD10, IR10, and DIST for various values of λ. The MIP models are excluded due to scalability issue [2]. We compare the previously used performance measures execution time, solution cardinalities ($\|z\|_0$), and solution quality (δ_{sol}). In addition, we also compare sensitivities (true positive ratio) of the solutions. By comparing with the known structure with 20 arcs, we calculate directed true

2. Although m is small, with n =7466 observations, the MIP models have at least $7466 \cdot 11 \cdot 2 = 164{,}252$ continuous variables and $7466 \cdot 11 = 82{,}126$ constraints just for the residual terms. Combined with the complexity increment due to the binary variables and acyclicity constraints, it was not feasible to obtain a reasonable solution by any of the MIP models.

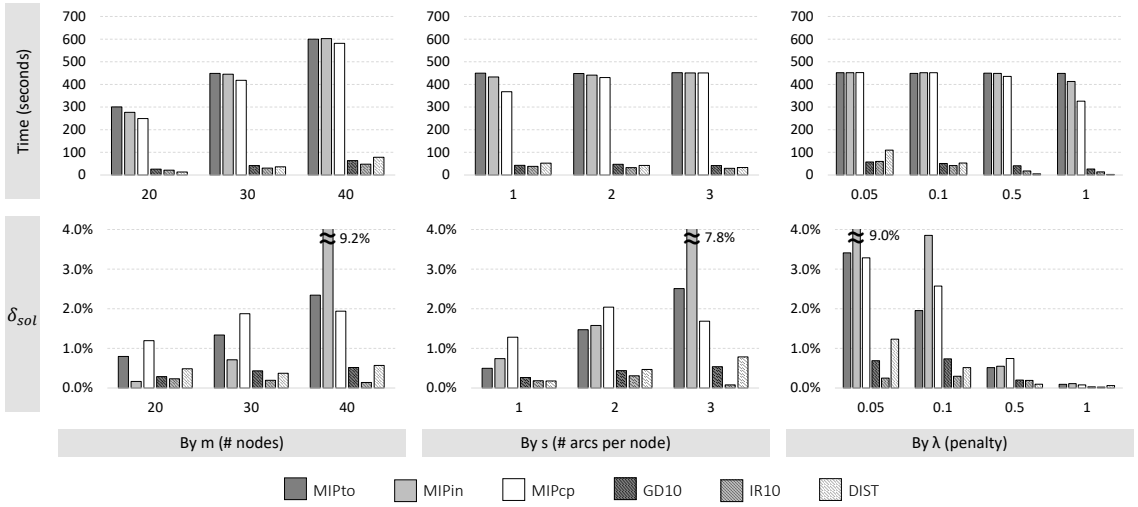

Figure 6: Performance of all models and algorithms (sparse data with $n = 100$ and $m \in \{20, 30, 40\}$)

positive (dTP) and undirected true positive (uTP). If arc (j, k) is in the known structure, dTP counts only if arc (j, k) is in the algorithm's solution, whereas uTP counts either of arcs (j, k) or (k, j) is in the algorithm's solution.

The solution times of the three algorithms are all within a few seconds, as m is small. Also, the solution cardinalities ($\|z\|_0$) are similar. The best δ_{sol} value among the three algorithms in each row is in boldface. We observe that GD10 provides the best solution (smallest δ_{sol}) in most cases and IR10 is the second best. The δ_{sol} values of DIST increase as λ increases. This is consistent with the findings in Section 5.1. Note that the density of the underlying structure is $20/11^2 = 0.165$, which is dense. This explains the good performance of GD10 and IR10. On the other hand, even though GD10 provides the best objective function values for most of the cases, the dTP and uTP values of GD10 are not always the best. The highest value among the three algorithms in each row is in boldface. While δ_{sol} values of DIST are the largest among the three algorithms, dTP and uTP values are the best in some cases. When λ is small, DIST tends to have higher dTP and uTP. However, as λ increases, GD10 gives the best dTP and uTP values. To further improve the prediction power, we may need weighting features or observations.

From Table 2, we observe that a slight change in solution quality δ_{sol} affects the final selection of DAG significantly. Also, the best objective function value does not necessarily give the highest true positive value since the L_1-norm penalized least square (14) may not be the best score function. In Figure 7, we present the graphs of known casual interactions (a), estimated subgraph by GD10 (b), IR10 (c), and DIST (d). All graphs are obtained with $\lambda = 0.25$, but the numbers of arcs are different (see Table 2). In fact, the difference between δ_{sol} values of IR10 and DIST is less than 1% while the subgraphs have only 10 common arcs.

	GD10					IR10					DIST				
λ	*time*	$\|z\|_0$	δ_{sol}	dTP	uTP	*time*	$\|z\|_0$	δ_{sol}	dTP	uTP	*time*	$\|z\|_0$	δ_{sol}	dTP	uTP
0.5	14.6	9	**0.00%**	0.22	**0.56**	10.8	9	0.22%	0.22	**0.56**	5.8	9	0.38%	**0.56**	**0.56**
0.45	13.7	11	**0.00%**	0.27	0.55	10.9	13	0.30%	0.15	0.46	8.5	9	0.49%	**0.56**	**0.56**
0.4	14.3	11	**0.00%**	0.27	**0.55**	14.8	13	0.36%	0.15	0.46	7.0	13	0.61%	**0.38**	0.46
0.35	16.0	13	**0.00%**	0.23	**0.54**	12.4	17	0.42%	0.12	0.41	8.6	15	0.73%	**0.33**	0.47
0.3	15.1	13	**0.00%**	0.23	0.46	14.3	17	0.21%	0.24	0.41	8.0	17	0.84%	**0.29**	**0.47**
0.25	14.4	16	**0.00%**	**0.38**	**0.56**	14.8	20	0.22%	0.25	0.50	6.7	20	0.94%	0.30	0.50
0.2	16.0	16	**0.00%**	0.31	**0.56**	14.6	22	0.31%	0.32	0.55	6.8	21	1.12%	**0.33**	0.52
0.15	16.4	21	**0.00%**	**0.33**	**0.57**	15.6	25	0.32%	0.32	0.52	9.5	23	1.26%	0.30	0.52
0.1	17.4	24	**0.00%**	0.25	**0.50**	11.0	28	0.26%	0.07	0.43	9.1	27	1.45%	**0.26**	0.44
0.05	17.7	28	0.32%	**0.39**	**0.50**	11.5	30	**0.00%**	0.10	0.47	10.1	32	1.87%	0.22	0.41

Table 2: Performance on the real data set from Sachs et al. (2005)

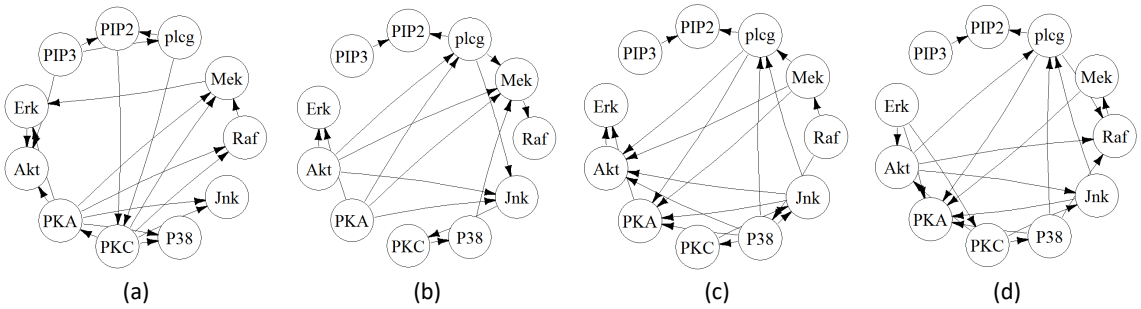

Figure 7: Known DAG (a) and estimated subgraphs with $\lambda = 0.25$ by GD10 (b), IR10 (c), and DIST (d)

6. Conclusion

We propose an MIP model and iterative algorithms based on topological order. Although the computational experiment is conducted for Gaussian Bayesian network learning, all the proposed model and algorithms are applicable for problems following the form in (1). While many MIP models and algorithms are designed based on arc search, using topological order provides some advantages that improve solution quality and algorithm efficiency.

1. DAG constraints (acyclicity constraints) are automatically satisfied when arcs from high order nodes to low order nodes are used.

2. In applying the concept for MIP, a lower number of constraints is needed ($O(m^2)$) whereas arc based modeling can have exponentially many constraints in the worst case.

3. In applying the concept in designing iterative algorithms, one of the biggest merits is the capability of utilizing the maximum number of arcs possible ($\frac{m(m-1)}{2}$), while arc based algorithms struggle with using all possible arcs.

The proposed MIP model has the smallest number of constraints while the number of binary variables is in the same order with the already known MIP models. It performs as good as a cutting plane algorithm. The proposed iterative algorithms get the biggest

benefit when the solution matrix is dense. The result presented in Section 5.1 clearly indicates that the topological order based algorithms outperform when the density of the resulting solution is high. On the other hand, arc-based search algorithms, represented by DIST in our experiment, can be efficient when the desired solutions are very sparse.

Comparing all models and algorithms used in the experiment, we observe that the MIP models are not competitive or scale well compared to the heuristic algorithms except for small instances. The experiment shows that the solution times of the MIP models are significantly affected by the number of nodes m. For Gaussian Bayesian network learning, we observe that large n could also decrease the MIP model efficiency even when m is small (Section 5.4). Among the iterative algorithms, we recommend DIST for very sparse high dimensional data and GD10 and IR10 for dense data. Among the two topological order based algorithms, GD10 performs slightly better and is more stable.

Appendix A. Greedy Algorithm for Projection Problem

In this section, we present the detail derivations and proofs of Algorithm 3 (*greedy*). The algorithm sequentially determines topological order by optimizing the projection problem given an already fixed order up to the iteration point. Solving the projection problem is $\mathcal{NP}$-complete and our algorithm may not give a global optimal solution. However, the result of this section shows that Algorithm 3 gives an optimal choice of the next node to have fixed order given pre-fixed orders.

We start by describing some properties of Y^* in the following three lemmas. For the following lemmas, let π_j^* represent the topological order of node j defined by Y^*.

Lemma 5 *For any Y_{jk}^*, we must have either $Y_{jk}^* = 0$ or $Y_{jk}^* = U_{jk}^t$.*

Proof For a contradiction, let us assume that there exist indices q and r such that $Y_{qr}^* \neq 0$ and $Y_{qr}^* \neq U_{qr}^t$. Let us create a new solution $\bar{Y}$ such that $\bar{Y} = Y^*$ except $\bar{Y}_{qr} = U_{qr}^t$. Note that $\bar{Y}$ is a feasible solution to (12) because $supp(\bar{Y}) \leq supp(Y^*)$ element-wise since $Y_{qr}^* \neq 0$. Further, we have $\|Y^* - U^t\|_2 > \|\bar{Y} - U^t\|_2$ because $(Y_{qr}^* - U_{qr}^t)^2 > 0 = (\bar{Y}_{qr} - U_{qr}^t)^2$ and $\bar{Y} = Y^*$ except $Y_{qr}^* \neq \bar{Y}_{qr}$. This contradicts optimality of Y^*. ■

Note that Lemma 5 implies that solving (12) is essentially choosing between 0 and U_{jk}^* for Y_{jk}^*. This selection is also based on the following property.

Lemma 6 *If $\pi_j^* > \pi_k^*$, then $Y_{jk}^* = U_{jk}^t$.*

Proof For a contradiction, let us assume that there exist indices q and r such that $Y_{qr}^* \neq U_{qr}^t$ while $\pi_q^* > \pi_r^*$. Let us create a new solution $\bar{Y}$ such that $\bar{Y} = Y^*$ except $\bar{Y}_{qr} = U_{qr}^t$.

1. If $Y_{qr}^* \neq 0$, then $\bar{Y}$ is a DAG since $supp(\bar{Y}) \leq supp(Y^*)$ element-wise.
2. If $Y_{qr}^* = 0$, then arc (q, r) can be used in the solution without creating a cycle because $\pi_q^* > \pi_r^*$. Hence, $\bar{Y}$ is a DAG.

Therefore, $\bar{Y}$ is a feasible solution to (12). However, it is easy to see that $\|\bar{Y} - U^t\|_2 < \|Y^* - U^*\|_2$ because $(Y_{qr}^* - U_{qr}^t)^2 > 0 = (\bar{Y}_{qr} - U_{qr}^t)^2$. This contradicts optimality of Y^*. ■

Given the topological order by Y^*, let $\hat{J}^k = \{j \in J^k | \pi_j^* > \pi_k^*\}$ be the subset of J^k such that the nodes in $\hat{J}^k$ are earlier than node k. Combining Lemmas 5 and 6, we conclude that Y^* has the following structure.

$$Y_{jk}^* = \begin{cases} U_{jk}^t & \text{if } j \in \hat{J}^k, \\ 0 & \text{if } j \in J \setminus \hat{J}^k, \end{cases} \qquad k \in J \tag{18}$$

Further, we can calculate node k's contribution to the objective function value without explicitly using Y^*.

Lemma 7 *For each node $k \in J$, it contributes*

$$\sum_{j \in J \setminus \hat{J}^k} (U_{jk}^t)^2$$

to the objective function value for (12)*. In other words, the contribution of node k is the squared sum of U_{jk}^t for nodes with $\pi_j^* < \pi_k^*$.*

Proof For node k, we can derive

$$\sum_{j\in J}(Y^*_{jk} - U^t_{jk})^2 = \sum_{j\in J\setminus \hat{J}^k}(Y^*_{jk} - U^t_{jk})^2 = \sum_{j\in J\setminus \hat{J}^k}(U^t_{jk})^2,$$

where both equal signs are due to (18). The first equality is due to $Y^*_{jk} = U^t_{jk}$ for $j \in \hat{J}^k$ and the second equality holds since $Y^*_{jk} = 0$ for $j \in J \setminus \hat{J}^k$. ■

We next detail the derivation of the greedy algorithm presented in Algorithm 3. Let $\bar{J} \subseteq J$ be the index set of yet-to-be-ordered nodes and $\bar{J}^c = J \setminus \bar{J}$ be the index set of the nodes that have already been ordered. The procedure is equivalent to iteratively solving

$$k^* = \text{argmin}_{k\in\bar{J}}\Big\{ \min_{Y_k} \big\{ \sum_{j\in J}(Y_{jk} - U^t_{jk})^2\big\}\Big\}, \tag{19}$$

where $Y_k = [Y_{1k}, Y_{2k}, \cdots, Y_{mk}] \in \mathbb{R}^{m\times 1}$ is the column in Y corresponding to node k. Set $\bar{J}$ is updated by $\bar{J} = \bar{J} \setminus \{k^*\}$ and $\pi^*_{k^*} = |\bar{J}|$ after solving (19). We propose an algorithm to solve (19) based on the properties of Y^* described in Lemmas 5 - 7. Given $\bar{J}$, we solve

$$k^* = \text{argmin}_{k\in\bar{J}}\Big\{ \sum_{j\in\bar{J}}(U^t_{jk})^2\Big\}. \tag{20}$$

Next, we show that solving (20) gives an optimal solution $\bar{Y}_{jk^*}$, $j \in J^k$ to (19). We can actually replicate the properties of Y^* for $\bar{Y}_{jk^*}$.

Lemma 8 *An optimal solution to* (19) *must have either* $\bar{Y}_{jk^*} = 0$ *or* $\bar{Y}_{jk^*} = U^t_{jk^*}$ *for all* $j \in J^k$.

Lemma 9 *An optimal solution to* (19) *must have* $\bar{Y}_{jk^*} = U^t_{jk^*}$ *for* $j \in J \setminus \bar{J}$.

Lemma 10 *An optimal solution to* (19) *must satisfy* $\sum_{j\in J}(\bar{Y}_{jk^*} - U^t_{jk^*})^2 = \sum_{j\in\bar{J}}(U^t_{jk^*})^2$

The proofs are omitted as they are similar to the proofs of Lemmas 5 - 7, respectively. Note that, by Lemma 10, we show the equivalence of $\sum_{j\in J}(Y_{jk^*} - U^t_{jk^*})^2$ and $\sum_{j\in\bar{J}}(U^t_{jk^*})^2$. This result also holds for each term, $k \in \bar{J}$, of the argmin function in (19). Hence, the following lemma holds.

Lemma 11 *Solving* (20) *is equivalent to solving* (19).

Observe that (20) is used in Line 3 of the greedy algorithm in Algorithm 3. Hence, by property (19), the algorithm gives an optimal choice of node to be fixed with the pre-fixed topological order.

Appendix B. Summary Statistics for Maximum Coefficients

In this section, we show that the heuristic formula (17) for selecting big M gives reasonable and large enough values. Note that, when creating a synthetic instance, we used a random DAG to generate multivariate data. Although the optimal DAG for the penalized least

squares is unknown, with appropriate penalty constants, we can use the implanted DAG to obtain coefficients estimation. That is, we calculate

$$\hat{B} = \max_{j \in J^k, k \in K} |\hat{\beta}_{jk}|$$

by using the implanted DAG. The $\hat{B}$ value is then compared with the big M value in (17) for all instances used for MIP models (sparse data with $n = 100$ and $m \in \{20, 30, 40\}$). We calculate minimum, average, and maximum of $\hat{B}$ and M by m, s, and λ. The result is presented in Table 3. Note that we have $\hat{B} < M$ for all columns Min, Avg, and Max and for all rows. Although only the summary statistics are presented in Table 3, we observed that M is greater than $\hat{B}$ for all cases considered.

By		$\hat{B}$			M		
Param	Value	Min	Avg	Max	Min	Avg	Max
m	20	0.17	0.65	1.48	0.33	1.34	3.73
	30	0.18	0.62	0.95	0.36	1.32	3.10
	40	0.18	0.63	1.03	0.36	1.33	3.30
s	1	0.17	0.53	0.79	0.33	1.13	2.04
	2	0.23	0.63	0.94	0.45	1.33	3.54
	3	0.32	0.74	1.48	0.63	1.53	3.73
λ	0.05	0.62	0.82	1.48	1.24	1.95	3.73
	0.1	0.60	0.78	0.96	1.19	1.54	2.35
	0.5	0.41	0.59	0.74	0.82	1.14	1.45
	1	0.17	0.35	0.49	0.33	0.69	0.97

Table 3: Comparison of maximum coefficients and big M (sparse data with $n = 100$ and $m \in \{20, 30, 40\}$)

References

Bryon Aragam and Qing Zhou. Concave penalized estimation of sparse Gaussian Bayesian networks. *Journal of Machine Learning Research*, 16:2273–2328, 2015.

Ali Baharev, Hermann Schichl, and Arnold Neumaier. An exact method for the minimum feedback arc set problem. *Technical report*, 2015. Available from http://www.mat.univie.ac.at/∼neum/.

G Bolotashvili, M Kovalev, and Eberhard Girlich. New facets of the linear ordering polytope. *SIAM Journal on Discrete Mathematics*, 12(3):326–336, 1999.

David Maxwell Chickering. Learning Bayesian networks is NP-complete. In *Learning from data*, pages 121–130. Springer, 1996.

David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3(Nov):507–554, 2002.

William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

Byron Ellis and Wing Hung Wong. Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482):778–789, 2008.

Guy Even, J Seffi Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

Henning Fernau and Daniel Raible. Exact algorithms for maximum acyclic subgraph on a superclass of cubic graphs. In *International Workshop on Algorithms and Computation*, pages 144–156. Springer, 2008.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical Lasso. *Biostatistics*, 9(3):432–441, 2008.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003.

Fei Fu and Qing Zhou. Learning sparse causal Gaussian networks with experimental intervention: regularization and coordinate descent. *Journal of the American Statistical Association*, 108(501):288–300, 2013.

Michel X Goemans and Leslie A Hall. The strongest facets of the acyclic subgraph polytope are unknown. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 415–429. Springer, 1996.

Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.

Martin Grötschel, Michael Jünger, and Gerhard Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28–42, 1985.

Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *49th Annual IEEE Symposium on Foundations of Computer Science*, pages 573–582. IEEE, 2008.

Sung Won Han, Gong Chen, Myun-Seok Cheon, and Hua Zhong. Estimation of directed acyclic graphs through two-stage adaptive Lasso for gene network inference. *Journal of the American Statistical Association*, 111(515):1004–1019, 2016.

Refael Hassin and Shlomi Rubinstein. Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51(3):133–140, 1994.

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

R Kaas. A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research*, 8(4):355–362, 1981.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47(11):1–26, 2012.

Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 95–103. ACM, 2007.

Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the mdl principle. *Computational Intelligence*, 10(3):269–293, 1994.

Janny Leung and Jon Lee. More facets from fences for linear ordering and acyclic subgraph polytopes. *Discrete Applied Mathematics*, 50(2):185–200, 1994.

CL Lucchesi and DH Younger. A minimax theorem for directed graphs. *Journal of the London Mathematical Society*, 17(3):369–374, 1978.

John E Mitchell and Brian Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In *High Performance Optimization*, pages 349–366. Springer, 2000.

Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Structure discovery in Bayesian networks by sampling partial orders. *The Journal of Machine Learning Research*, 17(1): 2002–2048, 2016.

Young Woong Park and Diego Klabjan. Subset selection for multiple linear regression via optimization. *Technical report*, 2013. Available from http://www.klabjan.dynresmanagement.com.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL `https://www.R-project.org/`.

Vijaya Ramachandran. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms*, 9(3):299–313, 1988.

Garvesh Raskutti and Caroline Uhler. Learning directed acyclic graphs based on sparsest permutations. *arXiv preprint arXiv:1307.0366*, 2013.

Karen Sachs, Omar Perez, Dana Pe'er, Douglas A. Lauffenburger, and Garry P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005.

Ali Shojaie and George Michailidis. Penalized likelihood methods for estimation of sparse high-dimensional directed acyclic graphs. *Biometrika*, 97(3):519–538, 2010.

Sara Van de Geer and Peter Bühlmann. L0-penalized maximum likelihood for sparse directed acyclic graphs. *The Annals of Statistics*, 41(2):536–567, 2013.