Learning Large DAGs by Combining Continuous Optimization and Feedback Arc Set Heuristics

Pierre Gillot, Pekka Parviainen

University of Bergen
HIB - Thormøhlens gate 55
Postboks 7803 5020 Bergen
Pierre.Gillot@uib.no, Pekka.Parviainen@uib.no

Abstract

Bayesian networks represent relations between variables using a directed acyclic graph (DAG). Learning the DAG is an NP-hard problem and exact learning algorithms are feasible only for small sets of variables. We propose two scalable heuristics for learning DAGs in the linear structural equation case. Our methods learn the DAG by alternating between unconstrained gradient descent-based step to optimize an objective function and solving a maximum acyclic subgraph problem to enforce acyclicity. Thanks to this decoupling, our methods scale up beyond thousands of variables.

Introduction

Bayesian networks are probabilistic graphical models that represent joint distributions among random variables. They consist of a structure which is a directed acyclic graph (DAG) representing conditional independencies and parameters that specify local conditional distributions.

Bayesian networks can handle both discrete and continuous variables. In this work, we concentrate on continuous variables. Specifically, we study linear structural equation models (SEMs) where the local conditional distribution in a node is a Gaussian whose mean is a linear combination of the values of its parents.

Traditionally, there are two main approaches for learning DAGs. In *constraint-based* approach (see, e.g., (Pearl 2000; Spirtes, Glymour, and Scheines 2000)), one performs conditional independence tests and tries to construct a DAG that expresses the same conditional independencies as the test results. We take the *score-based* approach (see, e.g., (Cooper and Herskovits 1992; Heckerman, Geiger, and Chickering 1995)) where one tries to find a DAG that maximizes a score. Typically, one uses decomposable scores, that is, the score of a DAG is a sum of local scores for each node-parent set pair. This leads to a combinatorial optimization problem where one picks a parent set for each node while satisfying the constraint that the resulting graph is acyclic.

The combinatorial learning problem is NP-hard (Chickering 1996) and developing scalable methods is challenging. Indeed, state-of-the-art exact learning methods scale only up to few hundred nodes (Cussens 2011) and scalable algorithms for SEMs rely on approaches such as local modifications (Aragam, Gu, and Zhou 2019). A recent breakthrough, NOTEARS (Zheng et al. 2018) circumvents the combinatorial problem by formulating a continuous acyclicity constraint. This enables usage of gradient-based optimization methods. The bottleneck with respect to scalability lies in the cubic complexity for the calculation of the acyclicity function which involves the computation of a matrix exponential. GOLEM (Ng, Ghassami, and Zhang 2020) is similar to NOTEARS but replaces the generalized LASSO objective found in NOTEARS by a log-likelihood-based fitness function. It shares however the same computational bottleneck as NOTEARS due to the acyclicity constraint. Some methods circumvent this bottleneck by finding a sparse graph without the acyclicity constraint and impose acyclicity afterwards (Varando 2020; Yu and Gao 2020).

Our goal is to develop a fast heuristic for learning DAGs in the linear SEM setting. In other words, we trade accuracy for speed. We speed-up learning by decoupling the optimization of the objective function from the acyclicity constraint in a similar fashion as (Park and Klabjan 2017)[1]. At a general level, we learn by iteratively repeating the following steps:

1. Given an acyclic graph, find a graph (possibly cyclic) which is better in terms of the objective function value.
2. Given a cyclic graph, find an acyclic graph.

The first step can be solved efficiently using state-of-the-art gradient-based solvers. We present two variants for this step. *ProxiMAS* uses proximal gradient descent whereas *OptiMAS* uses standard automatic differentiation and gradient-based updates.

In the second step, the cyclic solution from the first step is converted into an acyclic one. The quality of the final solution depends crucially on the quality of this conversion. We solve an instance of maximum acyclic subgraph (MAS) problem which has been previously used to learn DAGs (Gillot and Parviainen 2020). Intuitively, we prefer keeping arcs whose weights are far from zero. Solving the MAS problem exactly is NP-hard but there exists efficient heuristics for solving its complement, the feedback arc set (FAS) problem (Simpson, Srinivasan, and Thomo 2016).

[1]Differences are discussed in Section "Proposed method".

Our experiments show that our methods can quickly find reasonable solutions on datasets with thousands of variables and beyond, even with modest running time. OptiMAS and ProxiMAS perform well compared to GD, NOTEARS and GOLEM in large-scale learning when resources are limited in terms of processors and memory.

Background

Linear structural equation models and Bayesian network structure learning

A Bayesian network is a representation of a joint probability distribution. It consists of two parts: a structure and parameters. The structure of a Bayesian network is a DAG $G = (N, A)$ where N is the node set and A is the directed adjacency matrix; we denote the parent set of a node v by Pa_v. Parameters specify local conditional distributions $P(v \mid Pa_v)$ and the joint probability distribution is represented by a factorization

$$P(N) = \prod_{v \in N} P(v \mid Pa_v).$$

We consider linear structural equation models (SEMs) where local conditional distributions are Gaussian distributions whose mean is a linear combination of the values of the parents of the variable. The structure of a linear SEM is determined by a weights matrix $W \in \mathbb{R}^{d \times d}$; $W(i, j)$ is non-zero if and only if $A(i, j) = 1$, that is, there is an arc from i to j. For a d-dimensional data vector x, we have

$$x = xW + e,$$

where e is a d-dimensional noise vector. The elements of e are independent of each other.

The goal in Bayesian network structure learning is to find a DAG G that fits the data. We are given a data matrix $X \in \mathbb{R}^{n \times d}$ with n samples of d-dimensional vectors. Our goal is to find a weights matrix $W \in \mathbb{R}^{d \times d}$ that represents an acyclic graph. To quantify how well the DAG and the weights fit the data, we can use the least-squares loss $\frac{1}{2n}\|XW - X\|_2^2$. Furthermore, we want to induce sparsity and therefore we add a regularization term $g(W)$, which we require to be convex. This leads to the following optimization problem.

$$\begin{aligned} \underset{W}{\operatorname{argmin}} \ & \frac{1}{2n}\|XW - X\|_2^2 + \lambda_1 g(W) \\ \text{s.t.} \quad & W \text{ is acyclic.} \end{aligned} \tag{1}$$

In the above formulation, λ_1 is a user-defined constant that determines the strength of regularization. To induce sparsity, we regularize with $L1$-norm, that is, $g(W) = \|W\|_1 = \sum_{i,j} |W(i, j)|$.

Maximum acyclic subgraph and feedback arc set

Formally, the maximum acyclic subgraph (MAS) problem is defined as follows. We are given a directed graph $G = (V, E)$ and a weight function $w(e)$ that assigns a weight for each arc $e \in E$. The goal is to find an acyclic graph $G' = (V, E')$ such that $E' \subseteq E$ and $\sum_{e \in E'} w(e)$ is maximized.

The maximum acyclic subgraph problem has a dual (or complementary) problem: the feedback arc set (FAS) problem. In FAS, we are given a directed graph $G = (V, E)$ and a weight function $w(e)$ just like in MAS. The goal is to find an arc set E'' such that $G'' = (V, E \setminus E'')$ is acyclic and $\sum_{e \in E''} w(e)$ is minimized. It is well known that $E' = E \setminus E''$. Thus, MAS can be solved by first solving FAS and then performing a simple subtraction of sets to obtain the corresponding solution to MAS.

Both MAS and FAS are NP-hard (Karp 1972). Therefore, exact algorithms are intractable on large graphs. Fortunately, there exists fast heuristics for FAS (Berger and Shor 1990; Eades, Lin, and Smyth 1993; Simpson, Srinivasan, and Thomo 2016).

Proposed method

A critical difficulty in solving Equation 1 stems from a combination of two problems:

- The quadratic objective function for the linear SEM problem has at most nd^3 quadratic terms. Indeed, the quadratic expression $\|XW - X\|_2^2$ is a sum of $n \times d$ squared expressions $\big((XW)_{i,j} - X_{i,j}\big)^2$, where each $(XW)_{i,j}$ is a linear expression consisting of d terms. As X is a continuous data matrix, one can rarely simplify the quadratic objective function significantly.
- Enforcing acyclicity. Standard constraints lead to NP-hard combinatorial problems. In the continuous setting, a smooth function exists that encodes acyclicity but with a prohibitive cubic complexity.

The main contribution of this work therefore is to address these two concerns. First, we decompose the quadratic optimization problem into a sequence of easier subproblems using iterative optimization techniques. Second, we separate entirely the quadratic optimization from the acyclicity constraints. Acyclicity is enforced by solving a MAS task as a proxy. The outline of the proposed method is shown in Algorithm 1 which iteratively does the following steps at each iteration k:

1. A new objective function is created based on the acyclic solution W_{k-1} obtained at the end of the previous iteration, which penalizes the original linear SEM objective by the least-square term $\frac{\lambda_2}{2}\|W - W_{k-1}\|_2^2$.
2. An optimization step is performed on the MAS-penalized problem, leading to a new cyclic solution $\widetilde{W}_k$.
3. An acyclic projection W_k of the previously obtained cyclic solution $\widetilde{W}_k$ is extracted, based on the squared values of $\widetilde{W}_k$. Formally, we attempt to compute $W_k = \widetilde{W}_k \odot A_k$, where A_k is the solution of the following MAS problem:

$$\begin{aligned} A_k = \ & \underset{A}{\operatorname{argmax}} \sum_{i,j} |\widetilde{W}_k(i, j)|^2 A(i, j) \\ & \text{s.t.} \quad A \in \{0,1\}^{d \times d} \text{ is acyclic.} \end{aligned} \tag{2}$$

   As mentioned before, finding optimal solutions for MAS is usually too time consuming and one has to resort to heuristics. We use a vectorized version of the approximation algorithm by Eades (Eades, Lin, and Smyth 1993) to find the acyclic weighted adjacency matrix W_k (Algorithm 2).

Intuitively, steps 1-2 are designed such that the updated weights matrix $\widetilde{W}_k$ will be constrained to remain in the vicinity of the previously found acyclic solution W_{k-1} returned by the MAS heuristic. In that sense, we approximate the acyclicity function used in NOTEARS and GOLEM by a projection term toward acyclic solutions which is much easier to compute and differentiate. Step 3 aims to preserve edges that represent the most important dependencies. In other words, we want to keep the weights that are far from zero. Solving MAS using weights that are squares of the original weights is equivalent to minimizing $\|W_k - \widetilde{W}_k\|_2$, which corresponds to finding the acyclic solutions that are closest to the cyclic solutions returned by the iterative optimization process. By alternating between optimization steps and MAS extractions via the repetition of steps 1-3, we aim to navigate through the search space of the original linear SEM problem by "following the trail" of a sequence of dynamically generated acyclic solutions.

The GD algorithm introduced by Park and Klabjan (2017) follows a similar strategy. It proceeds by repeating the three following steps: 1) make a gradient step for the linear SEM objective, 2) project the current cyclic solution to its MAS solution and 3) fit the linear SEM problem constrained by the newly found acyclic structure; as an optional fourth step, when the progress is too small they resort to an order-swapping heuristic. The main difference compared to our method is that we do not perform steps 3 and 4. The GD algorithm is greedier than our method, since we do not attempt to optimize the parameters of every discovered acyclic structure. From a practical standpoint, at each step of the GD algorithm, d LASSO instances have to be solved which becomes intractable for large-scale structure learning. In comparison, by directly plugging in step 1 the MAS projections to the linear SEM objective as dynamically evolving penalization terms, our approach circumvents entirely the need to solve these LASSO instances.

Connection with online convex optimization

Perhaps surprisingly, the dynamic nature of the proposed optimization procedure is not particularly challenging to work with in practice. Algorithm 1 can, indeed, be seen as a special case of an online convex optimization (OCO) problem. In his seminal paper, Zinkevich (2003) introduces this framework which he defines as such:

- $F \subset \mathbb{R}^n$ is a feasible set (assumed bounded, closed and non-empty).
- $(c_k)_k$ is an infinite sequence of smooth convex functions from F to $\mathbb{R}$, with bounded gradients.
- At each step k, an element $x_k \in F$ is selected, then assigned the cost $c_k(x_k)$.

In OCO, the standard optimization error becomes ill-defined and one seeks to optimize instead the so-called regret defined as

$$\text{regret} = \sum_{k \leq K} c_k(x_k) - \min_{x \in F} \sum_{k \leq K} c_k(x).$$

Zinkevich was the first to extend the gradient descent algorithm to its online form. It is well known that assuming convexity of the c_k and boundedness of the gradients, online gradient descent achieves $\mathcal{O}(\sqrt{K})$ regret bound and this bound is improved to $\mathcal{O}\big(\log(K)\big)$ assuming strong convexity of the c_k (Hazan 2019). More general classes of OCO algorithms have been studied (Hu, Pan, and Kwok 2009; Zhao, Qiu, and Liu 2018), notably (accelerated) proximal gradient descent algorithms concerned about composite convex functions of the form $\phi_k = f_k + g$ where only the f_k are smooth. Improved regret bounds again hold assuming strong convexity of the ϕ_k.

The proposed method is therefore theoretically well behaved: by considering the functions $f_k : W \mapsto \frac{1}{2n}\|XW - X\|_2^2 + \frac{\lambda_2}{2}\|W - W_{k-1}\|_2^2$ and $\phi_k : W \mapsto f_k(W) + \lambda_1 g(W)$ (Algorithm 1 line 2), notice that every ϕ_k is λ_2-strongly convex since for each k, the function $W \mapsto \phi_k(W) - \frac{\lambda_2}{2}\|W - W_{k-1}\|_2^2 = \frac{1}{2n}\|XW - X\|_2^2 + \lambda_1 g(W)$ is convex; Algorithm 1 therefore inherits aforementioned regret bounds from the OCO setting assuming boundedness of the gradients.

Implementation details

We implemented two variants of the proposed method:

- The first implementation, **ProxiMAS**, is designed to take full advantage of the properties of the objective functions ϕ_k, owing to the decoupling with acyclicity. Recall that we have $f_k : W \mapsto \frac{1}{2n}\|XW - X\|_2^2 + \frac{\lambda_2}{2}\|W - W_{k-1}\|_2^2$ and $\phi_k : W \mapsto f_k(W) + \lambda_1 g(W)$, where g is convex and the f_k are smooth and convex. By smooth, we mean that every f_k is differentiable with its gradient defined as $\nabla f_k(W) = \frac{1}{n} X^t X(W - I) + \lambda_2(W - W_{k-1})$ and one can easily show that every f_k has a Lipschitz-continuous gradient with optimal Lipschitz constant L_k upper-bounded by $L = \frac{1}{n}\|X^t X + n\lambda_2 I\|_2$, a value that does not depend on k. We can therefore use a proximal gradient descent optimization scheme as a backbone for our implementation, hence the name ProxiMAS. In practice, we use the FISTA algorithm (Beck and Teboulle 2009), an accelerated proximal algorithm with $\mathcal{O}(\frac{1}{k^2})$ convergence rate (where k is the number of steps in an offline optimization setting). One should notice that the running time of ProxiMAS does not depend on the number of samples n, since the proximal updates depend only on the covariance matrix $X^t X \in \mathbb{R}^{d \times d}$ which can be pre-computed.
- The second implementation, **OptiMAS**, replaces the proximal gradient descent by gradient descent-like steps. The main interest in doing so is that automatic differentiation will handle the optimization using a generic gradient descent-based solver. Despite the linear SEM objective being non-differentiable when the regularization term is the $L1$ norm, automatic differentiation frameworks can in practice optimize such non-smooth objective. Thus, OptiMAS is agnostic to the choice of the optimizer. In principle, one can use any variant of gradient-based optimizers. In our implementation, we have used Adam (Kingma and Ba 2015) as a backbone for automatic differentiation.

We stress that both variants are taking full advantage of vectorization and are thus GPU accelerated, first because we lifted the need to solve a sequence of LASSO instances at

Algorithm 1: Proposed method

Require: Data $X \in \mathbb{R}^{n \times d}$, initialization $W_0 \in \mathbb{R}^{d \times d}$, number of iterations K, $\lambda_1 > 0$, $\lambda_2 > 0$, optimizer
Ensure: Approximate solution to Equation 1
for $1 \leq k \leq K$ **do**
  Define $f_k : W \mapsto \frac{1}{2n}\|XW - X\|_2^2 + \frac{\lambda_2}{2}\|W - W_{k-1}\|_2^2$
    $\phi_k : W \mapsto f_k(W) + \lambda_1 g(W)$
  Do a descent step on ϕ_k: $\widetilde{W}_k = \text{step}(\phi_k, \text{ optimizer})$
  Project updated weights to their MAS approximation: $W_k = \text{greedy\_MAS}(\widetilde{W}_k)$
end for
return W_K

Algorithm 2: Vectorized greedy MAS

Require: $\widetilde{W} \in \mathbb{R}^{d \times d}$
Ensure: Approximate solution to Equation 2
$\hat{W} = \widetilde{W} \odot \widetilde{W}$
scores $= \hat{W}.\text{sum}(\text{dim} = 0)$
order $= \text{zeros}(\text{size} = d)$
ub $= (d+1) \times \max(\text{scores})$
for $0 \leq i < d$ **do**
  node $=$ argmin(scores)
  order$[-(i+1)]$ = node
  scores[node] = ub
  scores = scores $- \hat{W}[\text{node}, :]$
end for
order^{-1} $=$ argsort(order)
$W = \text{triangle\_low}(\widetilde{W}[\text{order}, \text{order}])[\text{order}^{-1}, \text{order}^{-1}]$
return W

each step, second because the MAS heuristic (Algorithm 2) is efficiently vectorized and runs quasi-linearly with respect to the number of nodes d when a GPU is available.

Experiments

We now present our experimental pipeline. We choose to compare the proposed algorithms (ProxiMAS and OptiMAS) against an iterative method (GD (Park and Klabjan 2017)) and the current state-of-the-art methods for sparse linear SEM structure recovery (NOTEARS (Zheng et al. 2018) and GOLEM (Ng, Ghassami, and Zhang 2020)).

Data generation

We adopt a similar setup as in (Zheng et al. 2018; Ng, Ghassami, and Zhang 2020): we first generate random DAGs based on Erdős-Rényi ("ER") and scale-free ("SF") models. We consider three sparsity regimes: sampled DAGs have $k \times d$ edges, where d is the number of nodes and $k \in \{1, 2, 4\}$. Graphs are denoted by "ERk" or "SFk" depending on their graph model and sparsity. Then, we generate the weighted adjacency matrices W by assigning random weights uniformly sampled in the range $[-2, -0.5] \cup [0.5, 2]$. Finally, we generate samples X following the linear SEM model: $X = E(I - W)^{-1}$, where $E \in \mathbb{R}^{n \times d}$ represents n i.i.d. samples from either a Gaussian, exponential or Gumbel distribution in $\mathbb{R}^d$. For all distributions, we investigate both the equal variance ("EV") setting with scale 1.0 for all variables and the non-equal variance ("NV") setting where every variable has its scale sampled uniformly in the range $[0.5, 1.5]$. Unless stated otherwise, n samples are generated both for the training data and for the validation data, with $n \in \{1000, 10000\}$.

Metrics

In order to evaluate the performance of the different methods, we compute the false negative rate (FNR), false positive rate (FPR) and the normalized structural Hamming distance (SHD) between predicted and groundtruth adjacency matrices. We proceed similarly with the undirected adjacency matrices. The Gaussian negative log-likelihood is also computed on the validation data (unseen during training). Aforementioned metrics are extracted at different thresholding values of the predicted weights matrices. Different methods behave differently at a fixed thresholding value. For example, we observed in our large-scale tests with limited running time that, for any fixed threshold, OptiMAS tends to produce significantly sparser graphs than NOTEARS and GOLEM. Thus, OptiMAS has lower FPR and higher FNR. In order to get a general performance score independent from the choice of a thresholding value, we additionally consider the average precision score as implemented in the scikit-learn package. This metric is robust against strong class imbalance as it occurs in large-scale sparse structure recovery. For brevity, only a fraction of the figures are shown in this paper.

Implementation

The two proposed methods (ProxiMAS and OptiMAS) are implemented in pytorch 1.8. The GOLEM method comes in two variations GOLEM-EV and GOLEM-NV originally implemented in tensorflow. In order to streamline benchmarking these variations were re-implemented in pytorch. The tensorflow and pytorch implementations were compared at fixed seed and produce nearly identical results given the same data; speedwise, we found the difference between the two implementations to be insignificant for large scale graphs with thousands of nodes. The original implementation of NOTEARS relies on a L-BFGS-B solver implemented in scipy and as mentioned in (Ng, Ghassami, and Zhang 2020) it does not scale to large instances with thousands of variables, thus for fairness we re-implemented it in pytorch as well. The existing implementation of the GD algorithm is written in R and uses the highly optimized package glmnet, thus we did not alter the implementation. All methods have full GPU support, with the exception of GD which relies on the LASSO implementation from the glmnet package and is restricted to CPU. For equal comparison, all methods are tested in a multi-threaded setting, without GPU.

Hyperparameters

All the tested methods have a hyperparameter λ_1 to promote sparsity; an additional hyperparameter λ_2 exists for all

tested methods except GD to enforce "dagness" (see Table 1). The values of these two hyperparameters yield different behavior depending on the method. The chosen value of λ_1 for NOTEARS and the chosen values of λ_1 and λ_2 for GOLEM are those recommended by their authors. The original NOTEARS implementation is based on an augmented Lagrangian method and does not use the λ_2 hyperparameter. We added this hyperparameter to our pytorch implementation of NOTEARS the same way as in the GOLEM implementation. We do not claim to have performed any model selection, but chose values that worked well in our tests.

	λ_1	λ_2
OptiMAS	0.1	20.0
ProxiMAS	0.1	20.0
NOTEARS	0.1	5.0
GOLEM-EV/NV	0.02/0.002	5.0
GD	0.1	-

Table 1: Sparsity (λ_1) and dagness (λ_2) hyperparameters

Additionally, in all experiments ProxiMAS and OptiMAS are configured to start enforcing acyclicity after 50 minutes of solving time, whereas NOTEARS and GOLEM enforce acyclicity the entire time as in their original papers. As suggested in (Ng, Ghassami, and Zhang 2020), we warm-start GOLEM-NV with a solution returned by GOLEM-EV when working on NV-generated data: the first half of the allowed running time runs GOLEM-EV whereas the second half runs GOLEM-NV. Finally, the methods that rely on automatic differentiation (NOTEARS, GOLEM and OptiMAS) all use the Adam optimizer (Kingma and Ba 2015) with default learning rate 0.001 as in (Ng, Ghassami, and Zhang 2020).

Benchmarking pipeline

We present three different experiments to emphasize the advantageous scaling of the proposed methods comparatively to the state of the art. The experiments were run on a cluster with Intel Xeon-Gold 6138 2.0 GHz / 6230 2.1 GHz CPUs. The number of cores and amount of memory used in each experiment are shown in Tables 2, 3, 4.

Results

Parameters	Values
d	$1000, 5000$
k	$1, 2, 4$
n	$1000, 10000$
Graph type	ER, SF
Noise type	Gaussian, exponential, Gumbel
Scale type	EV, NV
Repetitions	10
Total instances	1440
CPU cores	4
Memory (GB)	16
Runtime (h)	1

Table 2: Experiment 1 parameters

In the first experiment (see Table 2), we generated data with different noise models. We show selected results in Figures 1, 2. OptiMAS and ProxiMAS outperform the bench-

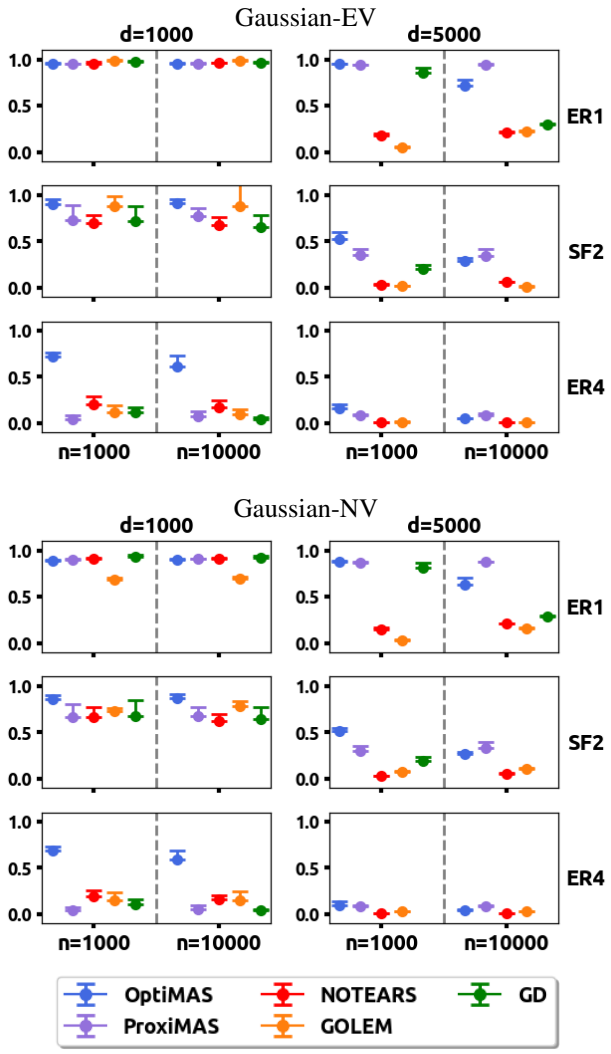

Figure 1: Mean average precisions for Gaussian noise distributions (EV and NV), d = number of nodes, n = number of samples. Confidence intervals show the standard deviation. Statistics are computed over 10 datasets.

mark methods in most instances, especially when $d = 5000$. Generally, GD performs equally good as OptiMAS and ProxiMAS when $d = 1000$ or $n = 1000$. However, it becomes slow when datasets grow. Especially, GD usually fails to even find a solution when there are lots of samples ($n = 10000$). Most of the time, NOTEARS and GOLEM are on par with GD or slightly better. Comparing OptiMAS and ProxiMAS, we notice that their performance is usually similar to each other. The main difference is that OptiMAS performs significantly better on more complex graphs (ER4) with $d = 1000$; we suspect this important disparity to be caused by numerical instabilities.

We also wanted to analyze how the available running time

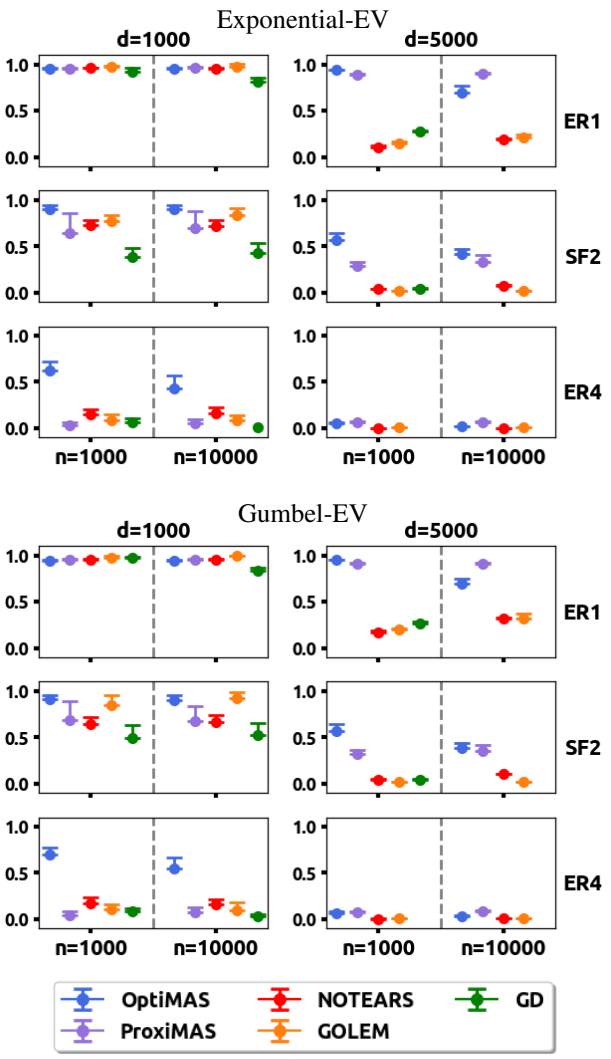

Figure 2: Mean average precisions for non-Gaussian noise distributions (EV), d = number of nodes, n = number of samples. Confidence intervals show the standard deviation. Statistics are computed over 10 datasets.

Parameters	Values
d	5000
k	1
n	1000, 10000
Graph type	ER
Noise type	Gaussian
Scale type	EV, NV
Repetitions	1
Total instances	4
CPU cores	4
Memory (GB)	16
Runtime (h)	24

Table 3: Experiment 2 parameters

affected each method. Therefore, we generated datasets from an ER1 model with 5000 nodes and Gaussian noise (see Table 3) and let all methods run for 24 hours. We recorded a snapshot of the weights matrix W every hour. Average precisions in the EV case are shown in Figure 3. We observe that

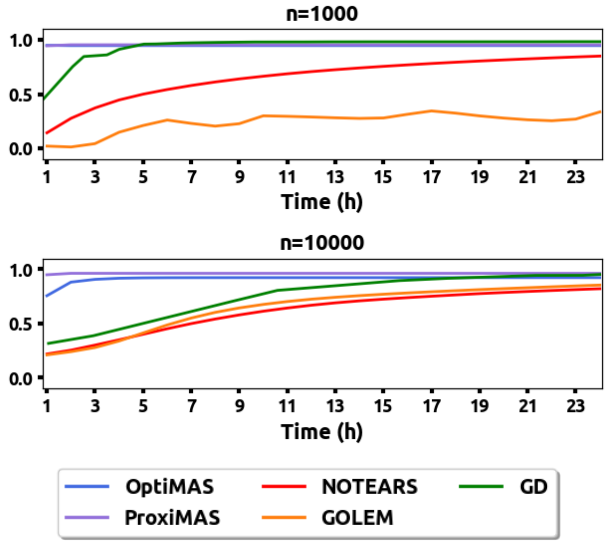

Figure 3: Average precision measured at different time points. Data generated from ER1 with 5000 nodes and Gaussian-EV noise. Note that on the top plot, the curves for OptiMAS and ProxiMAS are overlapping.

both OptiMAS and ProxiMAS find good solutions quickly. However, improvement after the first hour is negligible. GD starts slowly but eventually catches up with OptiMAS and ProxiMAS and often ends up with a slightly better solution. As in Experiment 1, we notice that the scalability of GD suffers from having lots of observations. Initially, NOTEARS is far behind but keeps improving significantly afterwards and after 24 hours it has found a solution that is almost as good as the ones found by OptiMAS and ProxiMAS. GOLEM performs similarly with NOTEARS when there are 10000 samples but struggles with 1000 samples.

Parameters	Values
d	5000, 10000, 15000, 20000
k	1
n	1000, 10000
Graph type	ER
Noise type	Gaussian
Scale type	EV
Repetitions	1
Total instances	8
CPU cores	32
Memory (GB)	128
Runtime (h)	1

Table 4: Experiment 3 parameters

Next, we study the scalability of the different methods. To

this end, we generated datasets from an ER1 model with varying number of nodes between 5000 and 20000 and Gaussian-EV noise (see Table 4). All methods were given 1 hour running time. Average precisions are shown in Figure 4. We

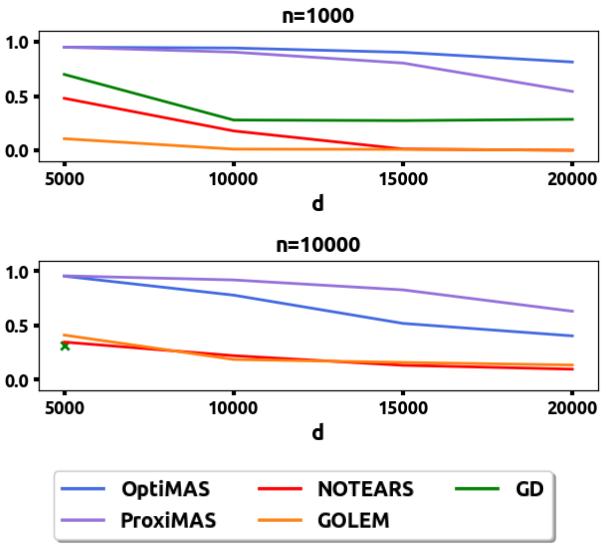

Figure 4: Scalability of different methods. Average precision is measured for different number of nodes d. Data generated from ER1 with Gaussian-EV noise.

notice that with 1000 samples the average precision for OptiMAS is high for all dataset sizes and decreases only little when the number of nodes grows. However, with 10000 samples average precision drops faster when the number of nodes grows. This may seem counter-intuitive as one would expect that more observations would lead to better performance. The likely explanation for this behavior is that, due to the fixed running time, OptiMAS performed fewer iterations and this countered the effect of increasing the number of observations at this scale. We can contrast this behavior to ProxiMAS whose running time does not depend on the number of observations. With $n = 1000$, ProxiMAS starts with nearly as high average precision as OptiMAS but its performance deteriorates quickly after 10000 nodes. However, with $n = 10000$, the drop is less significant and ProxiMAS clearly outperforms OptiMAS when there are 10000 or more nodes. We also notice that GOLEM and NOTEARS struggle to learn anything within an hour when there are more than 10000 nodes. GD performs better than GOLEM and NOTEARS when $n = 1000$ but when $n = 10000$ it only finds a solution for $d = 5000$ in the imparted time.

Table 5 shows rough memory usage of the different methods and their time per iteration (with acyclicity enforced for NOTEARS, GOLEM, OptiMAS and ProxiMAS) when the number of samples is small comparatively to the number of nodes ($n < d$). We see that in this regime, OptiMAS is the most time and memory efficient. ProxiMAS uses more time and memory than OptiMAS but much less than NOTEARS and GOLEM, while GD falls in between. Time per iteration for GD is very inconsistent due to the order-swapping heuristic it uses at certain iterations, thus we omitted it; as a rule, we observed that GD scales very unfavorably with respect to the number of samples especially.

	Memory (GB), Time/iteration (s)							
d	5000		10000		15000		20000	
OptiMAS	1	1	6	2	13	4	24	6
ProxiMAS	1	1	7	3	14	7	25	13
NOTEARS	6	6	23	40	52	100	92	250
GOLEM	6	6	23	45	53	120	94	280
GD	4	–	12	–	27	–	47	–

Table 5: Estimation of the memory usage and time per iteration (32 cores, ER1, Gaussian-EV, $n = 1000$)

Discussion

We presented two different heuristics (ProxiMAS and OptiMAS) for the structure recovery problem in the linear SEM case, revolving around a decoupling of the acyclicity constraints from the continuous optimization itself. We observed that both methods have excellent scaling (both space and time). OptiMAS scales particularly well when the number of samples n is small. On the contrary, ProxiMAS has invariant scaling with respect to n and scales in practice better than OptiMAS when the number of samples is large.

In our observations, both ProxiMAS and OptiMAS tend to get stuck on local extremum: the sequence of acyclic DAGs returned by the two methods is conditioned by the initial cyclic solution provided to them. This drawback can be alleviated by "warm-starting": run the algorithm initially without the MAS penalization and extraction steps (Algorithm 1 lines 2 and 4), then add these steps at some point during the execution. This strategy is made viable since a single MAS step is enough to guarantee acyclicity. Our experiments show that in practice, very good DAGs can be found even when most of the running time is dedicated to fitting the model without enforcing acyclicity.

Based on our experiments, OptiMAS and ProxiMAS are most competitive in situations where there is a large number of nodes and limited amount of computational resources. If there are a couple of thousands of nodes or less, the current state of the art is preferred. Similarly, if one can afford to run GD, NOTEARS or GOLEM for a long enough period of time, these algorithms will eventually outperform ProxiMAS/OptiMAS. However, in such a situation one could use OptiMAS or ProxiMAS to find an initial solution and use it to "warm-start" GD, NOTEARS or GOLEM.

Another limitation of our methods is that it is unclear at the moment how the theoretical results from online convex optimization translate with respect to the original problem. Currently, we are not aware of any necessary condition for local convergence of the proposed methods. This opens up an avenue of future research: Can we prove anything about the quality of the solutions? Can we say something for a specific type of data? Does the fact that we use a heuristic to find a maximum acyclic subgraph have an impact and would improving MAS also translate in better structure learning?

Acknowledgments

Parts of this work have been done in the context of CEDAS (Center for Data Science, University of Bergen, UiB).

The computations were performed on resources provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway.

The authors thank Young Woong Park for providing the R code for GD and Ignavier Ng for advice about using GOLEM.

References

Aragam, B.; Gu, J.; and Zhou, Q. 2019. Learning Large-Scale Bayesian Networks with the sparsebn Package. *Journal of Statistical Software*, 91(11).

Beck, A.; and Teboulle, M. 2009. A fast Iterative Shrinkage-Thresholding Algorithm with application to wavelet-based image deblurring. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 693–696.

Berger, B.; and Shor, P. W. 1990. Approximation Algorithms for the Maximum Acyclic Subgraph Problem. In *SODA'90*.

Chickering, D. M. 1996. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, 121–130. Springer-Verlag.

Cooper, G. F.; and Herskovits, E. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4): 309–347.

Cussens, J. 2011. Bayesian network learning with cutting planes. In *UAI*, 153–160. AUAI Press.

Eades, P.; Lin, X.; and Smyth, W. 1993. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6): 319 – 323.

Gillot, P.; and Parviainen, P. 2020. Scalable Bayesian Network Structure Learning via Maximum Acyclic Subgraph. In Jaeger, M.; and Nielsen, T. D., eds., *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, 209–220. PMLR.

Hazan, E. 2019. Introduction to Online Convex Optimization. *CoRR*, abs/1909.05207.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3): 197–243.

Hu, C.; Pan, W.; and Kwok, J. 2009. Accelerated Gradient Methods for Stochastic Optimization and Online Learning. In Bengio, Y.; Schuurmans, D.; Lafferty, J.; Williams, C.; and Culotta, A., eds., *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

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

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ng, I.; Ghassami, A.; and Zhang, K. 2020. On the Role of Sparsity and DAG Constraints for Learning Linear DAGs. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 17943–17954. Curran Associates, Inc.

Park, Y. W.; and Klabjan, D. 2017. Bayesian Network Learning via Topological Order. *Journal of Machine Learning Research*, 18: 99:1–99:32.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge university Press.

Simpson, M.; Srinivasan, V.; and Thomo, A. 2016. Efficient Computation of Feedback Arc Set at Web-Scale. *Proc. VLDB Endow.*, 10(3): 133–144.

Spirtes, P.; Glymour, C.; and Scheines, R. 2000. *Causation, Prediction, and Search*. Springer Verlag.

Varando, G. 2020. Learning DAGs without imposing acyclicity. *CoRR*, abs/2006.03005.

Yu, Y.; and Gao, T. 2020. DAGs with No Curl: Efficient DAG Structure Learning. In *Causal Discovery & Causality-Inspired Machine learning Workshop at 34th Conference on Neural Information Processing Systems*.

Zhao, Y.; Qiu, S.; and Liu, J. 2018. Proximal Online Gradient is Optimum for Dynamic Regret. *CoRR*, abs/1810.03594.

Zheng, X.; Aragam, B.; Ravikumar, P.; and Xing, E. P. 2018. DAGs with NO TEARS: Continuous Optimization for Structure Learning. *Advances in Neural Information Processing Systems (NeurIPS)*.

Zinkevich, M. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, 928–935. AAAI Press. ISBN 1577351894.