DAGMA: Learning DAGs via M-matrices and a Log-Determinant Acyclicity Characterization

Kevin Bello[†‡] **Bryon Aragam**[†] **Pradeep Ravikumar**[‡]
[†]Booth School of Business, The University of Chicago, Chicago, IL 60637
[‡]Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

The combinatorial problem of learning directed acyclic graphs (DAGs) from data was recently framed as a purely continuous optimization problem by leveraging a differentiable acyclicity characterization of DAGs based on the trace of a matrix exponential function. Existing acyclicity characterizations are based on the idea that powers of an adjacency matrix contain information about walks and cycles. In this work, we propose a new acyclicity characterization based on the log-determinant (log-det) function, which leverages the nilpotency property of DAGs. To deal with the inherent asymmetries of a DAG, we relate the domain of our log-det characterization to the set of *M-matrices*, which is a key difference to the classical log-det function defined over the cone of positive definite matrices. Similar to acyclicity functions previously proposed, our characterization is also exact and differentiable. However, when compared to existing characterizations, our log-det function: (1) Is better at detecting large cycles; (2) Has better-behaved gradients; and (3) Its runtime is in practice about an order of magnitude faster. From the optimization side, we drop the typically used augmented Lagrangian scheme and propose DAGMA (*Directed Acyclic Graphs via M-matrices for Acyclicity*), a method that resembles the central path for barrier methods. Each point in the central path of DAGMA is a solution to an unconstrained problem regularized by our log-det function, then we show that at the limit of the central path the solution is guaranteed to be a DAG. Finally, we provide extensive experiments for *linear* and *nonlinear* SEMs and show that our approach can reach large speed-ups and smaller structural Hamming distances against state-of-the-art methods.

1 Introduction

Structural equation models (SEMs) [43] are a standard modeling tool in several fields such as economics, social sciences, genetics, and causal inference, to name a few. Under this framework, in its general form, the value of each variable in the model is assigned by a general nonlinear, nonparametric function that takes as input the values of other variables in the model, thus, every SEM can be associated to a graphical model. In particular, we will consider graphical models that are directed acyclic graphs (DAGs).

A long-standing and active research area deals with the problem of learning the graphical structure (DAG) given passively observed data (a.k.a. causal discovery). Computationally, this problem is well-known to be NP-hard in general [10, 12], mainly due to the combinatorial nature of the space of DAGs. In this work, we will follow a score-based approach, a popular learning framework where the goal is to find a DAG that minimizes a *given* score [21, 11]. Recently, Zheng et al. [57] proposed an exact smooth nonconvex characterization of acyclicity which opened the door to solving the originally combinatorial problem via a set of tools that work in the fully continuous regime (e.g., gradient-based methods).

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Let $W \in \mathbb{R}^{d \times d}$ be a weighted adjacency matrix of a graph G of d nodes, and let $W \circ W$ denote the Hadamard product. The acyclicity function introduced by Zheng et al. [57] is defined as $h_{\text{expm}}(W) = \text{Tr}(e^{W \circ W}) - d$, where Tr denotes the trace of a matrix, and it was shown that $h_{\text{expm}}(W) = 0$ if and only if W corresponds to a DAG. A follow-up work [55] proposed another acyclicity function, which can be computed slightly faster, defined as $h_{\text{poly}}(W) = \text{Tr}((I + \frac{1}{d} W \circ W)^d) - d$, where I is the identity matrix. It was similarly shown by Yu et al. [55] that $h_{\text{poly}}(W) = 0$ if and only if W corresponds to a DAG. While seemingly different, both h_{expm} and h_{poly} are functions of the form $\text{Tr}(\sum_{k=0}^{d} c_p (W \circ W)^k) - d$ for some $c_k > 0$, as noted by Wei et al. [54]. To the best of our knowledge, all subsequent work [e.g., 58, 26, 59, 37, 36, 25, 40, to name a few] that has built upon the idea of using a continuous acyclicity characterization have used either h_{expm} or h_{poly}, or some acyclicity characterization in the form of a trace of a sum of matrix powers. The latter should come as no surprise, after all, a nonzero diagonal entry of the matrix power $(W \circ W)^k$ reveals the existence of a closed walk of length k in W.

Contributions. In this work, we propose a new acyclicity function that, as h_{expm} and h_{poly}, is both an exact and a smooth acyclicity characterization but that also possesses several advantages when compared to h_{expm} and h_{poly}. Specifically, we make the following set of contributions:

1. We propose a novel acyclicity characterization based on the log-determinant (log-det) function (see Theorem 1 and Section 3). In contrast to the classical log-det function defined over the cone of positive definite matrices, we define the domain of our log-det function to be the set of M-matrices due to the inherent asymmetries of DAGs. To our knowledge, we are the first to connect the notion of M-matrices to acyclicity and structure learning for DAGs.
2. We provide a detailed study of the properties of our log-det characterization in Section 3.1. First, we establish the similarities of our log-det function to other existing functions such as h_{expm} and h_{poly}. Second, we formally argue why these functions can be regarded as acyclicity regularizers, similar in spirit to the classical ℓ_1 and ℓ_2 regularizers. Third, we show that our log-det function is an *invex* function, i.e., all stationary points are global minimum, moreover, these stationary points correspond to DAGs.
3. In Section 3.2, we present three arguments as to why our log-det characterization could be preferred over other existing acyclicity functions. Briefly, our log-det function is better at detecting large cycles, has better behaved gradients, and can be computed in about an order of magnitude faster than h_{expm} and h_{poly}.
4. Motivated by the properties of our log-det function, in Section 4, we present DAGMA (*Directed Acyclic Graphs via M-matrices for Acyclicity*), a method that resembles the widely known central path approach for barrier methods [38]. We show that, at the limit of the central path, the solution is guaranteed to be a DAG. In contrast to the commonly adopted augmented Lagrangian scheme (originally proposed in [57]) each point in the central path of DAGMA is a solution to an unconstrained problem regularized by our log-det function.
5. Finally, in Section 5 and Appendix C, we provide extensive experiments for *linear* and *nonlinear* SEMs, where we show that DAGMA is capable of obtaining DAGs with *better accuracy*, i.e., lower structural Hamming distance (SHD), in a *much faster* way than the state-of-the-art.

1.1 Related work

The vast majority of methods for learning DAGs can be categorized into two groups: constraint-based algorithms, which rely on conditional independence tests; and score-based algorithms, which focus on finding a DAG that minimizes a given score/loss function. We briefly mention classical constraint-based methods as we follow a score-based approach. [49] developed the PC algorithm, a popular general method that learns the Markov equivalence class. Other algorithms such as [51, 29] are based on local Markov boundary search. Finally, hybrid approaches that combine constraint-based learning with score-based learning, such as [52, 15].

In the line of score-based methods, popular score functions include BDeu [21], BIC [31], and MDL [7]. Works that study linear Gaussian SEMs include [2, 3, 16, 17, 32, 44], and for linear non-Gaussian SEMs [27, 48]. For nonlinear SEMs, we note works on additive models [9, 13, 53], additive noise models [23, 45, 35], generalized linear models [42, 41, 19], and general nonlinear SEMs [34, 18].

More closely related to our work is the line of work built on the nonconvex continuous framework of Zheng et al. [57], such as, [58, 26, 59, 37, 36, 25, 40]. In contrast to our work, all of the aforementioned methods rely on the nonconvex acyclicity functions h_{expm} or h_{poly}, and with the exception of [37], all of these works also use the augmented Lagrangian scheme. Finally, the NoCurl method [56] also departs from using h_{expm}, although no other acyclicity constraint is proposed. Two immediate distinctions can be made to our work. First, we propose a novel acyclicity function based on the log-det function which we show to be prefereable to h_{expm} and h_{poly}. Second, we drop the commonly adopted augmented Lagrangian scheme to solve the constrained problem and instead follow a central path approach to leverage the barrier property of our log-det function.

Remark 1. *To avoid confusion, we also note that in the GOLEM method of Ng et al. [37] the score includes a log-determinant function of the form* $\log|\det(I-W)|$ *which stems from the* Gaussian log-likelihood*. While this expression is zero if W corresponds to a DAG, it is* ***not*** *an exact acyclicity characterization (i.e.* $\log|\det(I-W)|=0$ *does not imply W is a DAG). By contrast, the use of* M*-matrices in our work is crucial to translating the log-det function into a valid acyclicity regularizer. Moreover, it is not obvious how to extend GOLEM to arbitrary score functions, as their analysis is specific to the Gaussian likelihood function.*

2 Notation and background

Notation. We use $[d]$ to denote the set of integers $\{1\dots d\}$. For a square matrix A, we use $\lambda_i(A)$ to denote its i-th minimum eigenvalue, and use $\rho(A)$ to denote its spectral radius. Also, we use $\mathrm{Tr}(A)$, and $\det(A)$ to denote the trace and determinant of A. For matrices A, B, we let $A\circ B$ represent the element-wise or Hadamard product, moreover, the expression $A\geq B$ is entrywise, i.e., $A_{i,j}\geq B_{i,j}$. Then, we say that a matrix A is nonnegative whenever $A\geq 0$. For a complex number $a+bi$, we let $\Re(a+bi)=a$ denote its real part. We use $\|\cdot\|_p$ to denote the vector ℓ_p-norm, and $\|\cdot\|_{L^p}$ is the L^p-norm on functions. Lastly, $i\to j$ and $i\rightsquigarrow j$ represent an edge from i to j and a directed walk from i to j, respectively.

Let $X=(X_1,\dots,X_d)$ be a d-dimensional random vector. In its general form, a (nonparametric) structural equation model (SEM) consists of a set of equations of the form:

$$X_j = f_j(X, Z_j),\ \forall j\in[d], \tag{1}$$

where each $f_j:\mathbb{R}^{d+1}\to\mathbb{R}$ is a nonlinear nonparametric function, and Z_j is an exogenous variable representing errors due to omitted factors. We consider the Markovian model, which assumes that each Z_j is an independent random variable. Note that each f_j depends only on a subset of X (i.e., the parents of X_j) and Z_j; nonetheless, to simplify notation we ensure that each f_j is defined on the same space. Then $f=(f_1,\dots,f_d)$ induces a graphical structure, where we focus on directed acyclic graphs.

For any joint distribution over $Z=(Z_1,\dots,Z_d)$, the functions f_j define a joint distribution $\mathbb{P}(X)$ over the observed data, and a graph $G(f)$ via the dependencies in each f_j. Then, our goal is to learn $G(f)$ given n i.i.d. samples from $\mathbb{P}(X)$. In score-based learning, given a data matrix $\boldsymbol{X}=[\boldsymbol{x}_1,\dots,\boldsymbol{x}_d]\in\mathbb{R}^{n\times d}$, we define a score function $Q(f;\boldsymbol{X})$ to measure the 'quality' of a candidate SEM as follows: $Q(f;\boldsymbol{X})=\sum_{j=1}^d \mathsf{loss}(\boldsymbol{x}_j, f_j(\boldsymbol{X}))$, where we adopt the convention that $f_j(\boldsymbol{X})\in\mathbb{R}^n$. Here loss can be any loss function such as least squares $\mathsf{loss}(\boldsymbol{u},\boldsymbol{v})=\frac{1}{n}\sum_{i=1}^n(u_i-v_i)^2$ or the log-likelihood function, often augmented with a penalty such as BIC or ℓ_1. Given the score function Q and a family of functions $\mathcal{F}$, we seek to find the $f\in\mathcal{F}$ that minimizes the score, i.e.,

$$\min_{f\in\mathcal{F}} Q(f;\boldsymbol{X}) \quad \text{subject to} \quad G(f)\in\mathsf{DAGs}. \tag{2}$$

Similar to [58], we consider that each f_j lives in a Sobolev space of square-integrable functions whose derivatives are also square integrable. Then, let $\partial_k f_j$ denote the partial derivative of f_j w.r.t. X_k, it is easy to see that f_j is independent of X_k if and only if $\|\partial_k f_j\|_{L^2}=0$. With this observation, we construct the matrix $W(f)\in\mathbb{R}^{d\times d}$ with entries $[W(f)]_{i,j}\stackrel{\text{def}}{=}\|\partial_i f_j\|_{L^2}$, which precisely encodes the graphical structure amongst the variables X_j. That is $G(f)\in\mathsf{DAGs}\iff W(f)\in\mathsf{DAGs}$, where W is interpreted as the usual weighted adjacency matrix.

In practice, f is replaced with a flexible family of parametrized functions such as deep neural networks, so that problem (2) is finite dimensional. Finally, note that model (1) includes several

models as special cases, e.g., additive noise models, generalized linear models, additive models, polynomial regression, and index models. Previous work has studied the identifiability of several of these models, e.g., [23, 48, 45, 42, 41, 27]. In the sequel, we assume that the model is chosen such that the graph $G(f)$ is uniquely defined from (2).

3 A new characterization of acyclicity via log-determinant and M-matrices

In this section, we present our acyclicity characterization and study its properties. To declutter notation, in this section we simply write W instead of $W(f)$ to denote the *weighted* adjacency matrix of a graph; however, it should be clear that W depends on functions f_j as explained in the previous section.

We develop our characterization by first noting that for any weighted adjacency matrix W, we have that $W \in \mathsf{DAGs}$ if and only if W is a nilpotent matrix, or equivalently, all the eigenvalues of W are zero, i.e., $\lambda_i(W) = 0, \forall i \in [d]$. We then have the following obvious implications:

$$W \in \mathsf{DAGs} \iff (W \circ W) \in \mathsf{DAGs} \iff s - \lambda_i(W \circ W) = s, \forall i \in [d], \forall s \in \mathbb{R} \tag{3}$$

$$\implies \prod_{i=1}^{d} s - \lambda_i(W \circ W) = \det(sI - W \circ W) = s^d. \tag{4}$$

Implication (4) can be thought of as a relaxation of acyclicity in the sense that all DAGs satisfy (4), but not all W that satisfy (4) are DAGs. For example, let $s = 1$ and $W \circ W = \left[\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right]$, then it is clear that $\det(sI - W \circ W) = 1$ and, thus, (4) is satisfied; however, clearly W is not a DAG.

Thus, two immediate questions arise: (i) *Does there exist a domain for W such that (4) $\implies$ (3)?* (ii) *If so, what is the description of such domain?* We answer (i) in the affirmative, and answer (ii) by relating the domain of W to the set of M-matrices, which is defined below.

Definition 1 (M-matrix[1])**.** *An M-matrix is a matrix $A \in \mathbb{R}^{d\times d}$ of the form $A = sI - B$, where $B \geq 0$ and $s > \rho(B)$.*

M-matrices were introduced by Ostrowski [39] and arise in a variety of areas including input-output analysis in economics, linear complementarity problems in operations research, finite difference methods for partial differential equations, and Markov chains in stochastic processes. To the best of our knowledge, we are the first to connect the notion of M-matrices to graphical model structure learning through an acyclicity characterization.

The following proposition is an immediate consequence of Definition 1:

Proposition 1 (Berman and Plemmons [5])**.** *Let $A \in \mathbb{R}^{d\times d}$ be an M-matrix, then:*

(i) $\Re(\lambda_i(A)) > 0$, for all $i \in [d]$. *(ii) A^{-1} exists and is nonnegative, i.e., $A^{-1} \geq 0$.*

In the above, item (i) states that the eigenvalues of an M-matrix lie in the open right-half plane. Matrices which satisfy the latter property are also known as positive stable matrices. We thus have that M-matrices are special cases of positive stable matrices. It follows that the determinant of any M-matrix is positive.[2] This fact will be used for defining $h^s_{\mathrm{ldet}}(W)$, our acyclicity characterization given in Theorem 1. Finally, the nonnegativity of the inverse from item (ii) will be used to understand some properties of the gradient of $h^s_{\mathrm{ldet}}(W)$.

We now define the domain over which (4) $\implies$ (3) (see Theorem 1). For any $s > 0$, define

$$\mathbb{W}^s = \{W \in \mathbb{R}^{d\times d} \mid s > \rho(W \circ W)\}, \tag{5}$$

i.e. $\mathbb{W}^s$ is the set of real matrices whose entry-wise square given by $W \circ W$ have spectral radius less than s. The following lemma lists some relevant properties of $\mathbb{W}^s$.

[1] More precisely, we consider the definition of a non-singular M-matrix, which is sufficient for the purposes of this work.

[2] Note that due to asymmetries it is possible for an M-matrix to have complex eigenvalues. However, since we work with matrices with real entries, the complex eigenvalues come in conjugate pairs.

Lemma 1. *Let $\mathbb{W}^s$ defined as in (5). Then, for all $s > 0$:*

(i) DAGs $\subset \mathbb{W}^s$. *(ii) $\mathbb{W}^s$ is path-connected.* *(iii) $\mathbb{W}^s \subset \mathbb{W}^t$ for any $t > s$.*

In Lemma 1, item (i) implies that the W we look for is in the interior of $\mathbb{W}^s$; item (ii) indicates that we can find a path from any point in $\mathbb{W}^s$ to any DAG without leaving the set $\mathbb{W}^s$; item (iii) shows that one can vary s to enlarge or shrink the set $\mathbb{W}^s$.

Having defined the domain set $\mathbb{W}^s$, we now define our acyclicity characterization $h^s_{\text{ldet}} : \mathbb{W}^s \to \mathbb{R}$. Recall that item (i) in Proposition 1 implies that applying the logarithm function to the determinant of an M-matrix is always well defined, which motivates our following result.

Theorem 1 (Log-determinant characterization)**.** *Let $s > 0$ and let $h^s_{\text{ldet}} : \mathbb{W}^s \to \mathbb{R}$ be defined as $h^s_{\text{ldet}}(W) \stackrel{\text{def}}{=} -\log\det(sI - W \circ W) + d \log s$. Then, the following holds:*

(i) $h^s_{\text{ldet}}(W) \geq 0$, *with* $h^s_{\text{ldet}}(W) = 0$ if and only if W *is a DAG.*

(ii) $\nabla h^s_{\text{ldet}}(W) = 2(sI - W \circ W)^{-\top} \circ W$, *with* $\nabla h^s_{\text{ldet}}(W) = 0$ if and only if W *is a DAG.*

3.1 Properties of $h^s_{\text{ldet}}(W)$

In this section, we list several properties of our acyclicity characterization. The first property we discuss is related to the entries of $\nabla h^s_{\text{ldet}}(W)$.

Lemma 2. *For all $i, j \in [d]$, $[\nabla h^s_{\text{ldet}}(W)]_{i,j} = 0$ if and only if $W_{i,j} = 0$ or there is no directed walk from j to i. Equivalently, $[\nabla h^s_{\text{ldet}}(W)]_{i,j} \neq 0$ if and only if the edge $i \to j$ is part of some cycle in W. Finally, whenever $[\nabla h^s_{\text{ldet}}(W)]_{i,j} \neq 0$, we have that $\text{sign}([\nabla h^s_{\text{ldet}}(W)]_{i,j}) = \text{sign}(W_{i,j})$.*

The lemma above characterizes the nonzero entries and their signs of the gradient of h^s_{ldet}. This property formally offers a *regularizer* perspective for h^s_{ldet}, which we highlight next.

Remark 2 (A regularizer viewpoint)**.** *The function h^s_{ldet} promotes small parameters values in much the same way that the classical ℓ_1 and ℓ_2 regularizers do. In contrast to the latter regularizers, h^s_{ldet} will only shrink the value of a parameter $W_{i,j}$ if and only if the edge (i, j) is part of some cycle in W, as prescribed by Lemma 2.*

Recall that $h_{\text{expm}}(W) = \text{Tr}(e^{W \circ W}) - d$ and $h_{\text{poly}}(W) = \text{Tr}((I + \frac{1}{d} W \circ W)^d) - d$. It was noted by [54] that acyclicity characterizations of the form $\text{Tr}(\sum_{p=1}^d c_p (W \circ W)^d)$ for $c_p > 0$ also have the property in Lemma 2. This implies that h_{expm} and h_{poly} hold the property above and can also be interpreted as acyclicity regularizers. We note that the interesting part here is that h^s_{ldet} holds this property besides being different in nature to h_{expm} and h_{poly}.

Next, we state an important consequence of Lemma 2, which is related to the direction of $\nabla h^s_{\text{ldet}}(W)$.

Corollary 1. *At any $W \in \mathbb{W}^s$, the negative gradient $\nabla h^s_{\text{ldet}}(W)$ points towards the interior of $\mathbb{W}^s$.*

In optimization, the Hessian matrix plays an important role as it contains relevant information about saddle points and local extrema of a function, and is key to Newton-type methods. Another appealing property of $h^s_{\text{ldet}}(W)$ is that it has a Hessian described by a simple closed-form expression.

Lemma 3. *The Hessian of $h^s_{\text{ldet}}(W)$, which resides in $\mathbb{R}^{d^2 \times d^2}$, is given by:*

$$\nabla^2 h^s_{\text{ldet}}(W) = 4\,\text{Diag}(\text{vec}(W))(N \otimes N^\top)\,\text{Diag}(\text{vec}(W^\top))K^{dd} + 2\,\text{Diag}(\text{vec}(N^\top)),$$

where $N = (sI - W \circ W)^{-1}$, $\otimes$ denotes the Kronecker product, and K^{dd} is the $d^2 \times d^2$ commutation matrix such that $K^{dd}\,\text{vec}(A) = \text{vec}(A^\top)$, for any $d \times d$ matrix A.

Here we note that among $h_{\text{expm}}, h_{\text{poly}}$ and h^s_{ldet}, only h^s_{ldet} has a *tractable* expression for the Hessian. Furthermore, note that $\nabla^2 h^s_{\text{ldet}}(W)$ is indexed by vertex pairs so that the entry $[\nabla^2 h^s_{\text{ldet}}(W)]_{(k,l),(p,q)}$ corresponds to the second partial derivative $\frac{\partial^2 h^s_{\text{ldet}}}{\partial W_{k,l} \partial W_{p,q}}$. Using Lemma 3, we can characterize the nonzero entries, and their signs, of the Hessian of h^s_{ldet}.

Corollary 2. *The entries of the Hessian $\nabla^2 h^s_{\text{ldet}}(W)$ are described as follows:*

$$\left[\nabla^2 h^s_{\text{ldet}}(W)\right]_{(k,l),(p,q)} = \begin{cases} 4W_{l,k}N_{k,q}N_{p,l}W_{q,p} & \text{if } (k,l) \neq (p,q), \\ 4(W_{l,k})^2(N_{k,l})^2 + 2N_{k,l} & \text{if } (k,l) = (p,q), \end{cases}$$

where $N = (sI - W \circ W)^{-1}$. Moreover, an off-diagonal entry $[\nabla^2 h^s_{\text{ldet}}(W)]_{(k,l),(p,q)}$ is nonzero if and only if there exists a cycle in W of the form $q \to p \rightsquigarrow l \to k \rightsquigarrow q$, and has a sign equal to $\text{sign}(W_{l,k}W_{q,p})$. *Lastly, a diagonal entry $[\nabla^2 h^s_{\text{ldet}}(W)]_{(k,l),(k,l)}$ is nonzero if and only if there exists a directed walk from k to l, and its sign is always positive.*

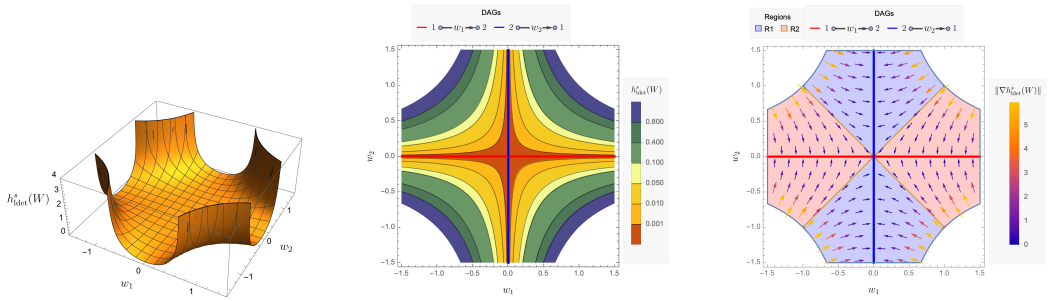
Recall from Theorem 1 that all DAGs attain the minimum value and are critical points of h^s_{ldet}; thus, DAGs are local (and global) minimum of h^s_{ldet} and the Hessian matrix evaluated at a DAG must be positive (semi)definite. Let us corroborate the latter, when W is a DAG, from Corollary 2 we have that the off-diagonal elements of $\nabla^2 h^s_{\text{ldet}}(W)$ are zero, while the diagonal entries are nonnegative. That is, the Hessian is positive semidefinite whenever W is a DAG. This implies that all its stationary points are global minima: Such functions are called *invex* [20, 30].

Corollary 3. *Let $s > 0$. Then, $h^s_{\text{ldet}}(W)$ is an invex function, i.e., all its stationary points are global minima, and these correspond to DAGs.*

We note that even though h_{expm} and h_{poly} are also invex functions, they were not explicitly considered as invex functions before. In fact, it was noted in [57] that DAGs were global minima of h_{expm} but no characterization of its stationary points were given. Wei et al. [54] noted that DAGs were stationary points of h_{expm} and h_{poly} but the notion of invexity was not explicitly stated.

Remark 3 (A dynamical system perspective)**.** *The importance of invexity here is that in the eyes of h^s_{ldet}, all DAGs are the same. That is, DAGs correspond to the set of attractors in h^s_{ldet}, and depending on the initial condition, the system will converge to a different attractor. This offers the following viewpoint for the role of the score function $Q(f; \boldsymbol{X})$, namely, "use the score Q to find a basin of attraction such that the force field of h^s_{ldet} will dictate the trajectory towards a DAG that is equal or close to the ground-truth".*

In Figure 1, we illustrate in a toy example the properties discussed in this subsection.



(a) $h^{s=1}_{\text{ldet}}(W)$ (b) Contours of $h^{s=1}_{\text{ldet}}(W)$ (c) Vector field of $\nabla h^{s=1}_{\text{ldet}}(W)$

Figure 1: Behavior of h^s_{ldet} for $W = \begin{bmatrix} 0 & w_1 \\ w_2 & 0 \end{bmatrix}$. Here clearly W is a DAG whenever one of w_1 or w_2 (or both) are zero. In particular, for (c) we note the perspective given in Remark 3, i.e., starting at any point in region R2 will converge to attractors (DAGs) of the form $X_1 \to X_2$ (red line); while starting at any point in region R1 will converge to attractors (DAGs) of the form $X_2 \to X_1$ (blue line).

3.2 Why the log-determinant regularizer is preferable to existing acyclicity regularizers

In this section, we present three arguments as to why one should use h^s_{ldet} instead of existing functions such as h_{expm} and h_{poly}. We invite the reader to look at Appendix B for additional details.

Argument (i). h^s_{ldet} does not diminish cycles of any length. Let us expand the functions h_{expm} and h_{poly} in their sum of matrix powers form, that is, $h_{\text{expm}}(W) = \sum_{k=0}^{\infty} {}^1\!/_{k!} \operatorname{Tr}((W \circ W)^k) - d$ and $h_{\text{poly}}(W) = \sum_{k=0}^{d} \binom{d}{k}/d^k \operatorname{Tr}((W \circ W)^k) - d$. Recall also that the entry $[(W \circ W)^k]_{i,i}$ represents the sum of weighted walks from node i to node i of length k, where each edge has weight $w^2_{u,v}$. Thus, one can notice that if W has cycles of length k, their contribution to h_{expm} and h_{poly} are *diminished* by ${}^1\!/_{k!}$ and $\binom{d}{k}/d^k$, respectively. Numerically, the latter can be problematic for the following reason: Cycles of length k can go undetected even for small values of d and k. In practice, a value of $h_{\text{expm}}, h_{\text{poly}} \in [10^{-8}, 10^{-5}]$ is typically regarded as zero [54, 57]. Consider a cycle graph of d nodes

where each edge weight is $+1$ or -1.[3] The plot in Figure 2 shows how the values of h_{expm} and h_{poly} decay much faster than that of h^s_{ldet}; in fact, at $d = 13$ we already observe $h_{\text{expm}}(W) \approx 10^{-9}$ and $h_{\text{poly}}(W) \approx 10^{-14}$, i.e., cycles of length at least 13 would be numerically undetected by h_{expm} and h_{poly}. In contrast, we observe that the value of the log-det function remains bounded away from zero and is able to detect larger cycles.

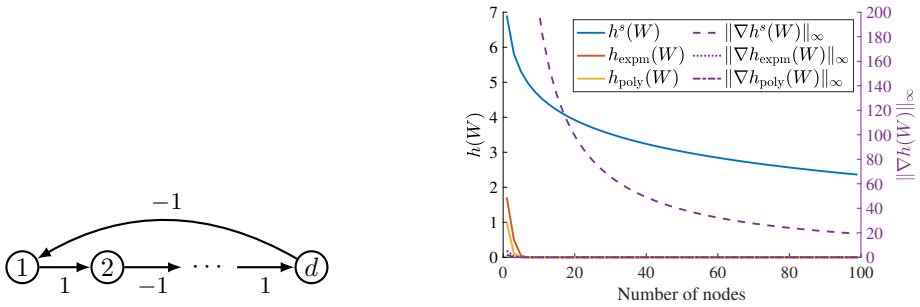


Figure 2: The values of h_{expm} and h_{poly} get very close to zero for a number of nodes as small as ten. In contrast, letting $s = 1.001$, we observe that h^s_{ldet} can stay away from zero even for a cycle graph of 100 nodes. Finally, we note a similar pattern for the entrywise ℓ_∞ norm of the gradients.

At first glance, it might seem difficult to directly compare the value of h^s_{ldet} to h_{expm} and h_{poly}. We next show that when $s = 1$, $h^{s=1}_{\text{ldet}}$ is an upper bound to h_{expm} and h_{poly}.

Lemma 4. *For all $W \in \mathbb{W}^{s=1}$, we have $h_{\text{poly}}(W) \leq h_{\text{expm}}(W) \leq h^{s=1}_{\text{ldet}}(W)$.*

The lemma above shows that in spite of $h^{s=1}_{\text{ldet}}$, h_{expm}, and h_{poly} being exact acyclicity characterizations, $h^{s=1}_{\text{ldet}}$ will attain the largest value.

Argument (ii). h^s_{ldet} has better behaved gradients. Similar to argument (i), we show in Appendix B that h_{expm} and h_{poly} are susceptible to vanishing gradients even when the graph contains cycles (see Figure 2). The following lemma states that the magnitude of each entry of ∇h^s_{ldet} at least as large as the magnitude of the corresponding entry of ∇h_{expm} and ∇h_{poly}, and hence, h_{ldet} has larger gradients to guide optimization.

Lemma 5. *For any walk of length k, its contribution to the gradients $\nabla h_{\text{expm}}(W)$ and $\nabla h_{\text{poly}}(W)$ are diminished by $1/k!$ and $\binom{d-1}{k}/(d-1)^k$, respectively. In contrast, $\nabla h^{s=1}_{\text{ldet}}(W)$ does not diminish any walk of any length. This implies that $|\nabla h_{\text{poly}}(W)| \leq |\nabla h_{\text{expm}}(W)| \leq |\nabla h^{s=1}_{\text{ldet}}(W)|$.*

Argument (iii). Computing h^s_{ldet} and ∇h^s_{ldet} is empirically faster. Even though h^s_{ldet}, h_{expm}, and h_{poly} all three share the same computational complexity of $\mathcal{O}(d^3)$, in practice h^s_{ldet} can be computed in about an order of magnitude faster than h_{expm} and h_{poly}. In Figure 3, we compare the runtimes of h^s_{ldet}, h_{expm} and h_{poly} for randomly generated matrices, where we observe that computing h^s_{ldet} can be 10x faster than h_{expm} and h_{poly}. See Appendix B for further details.

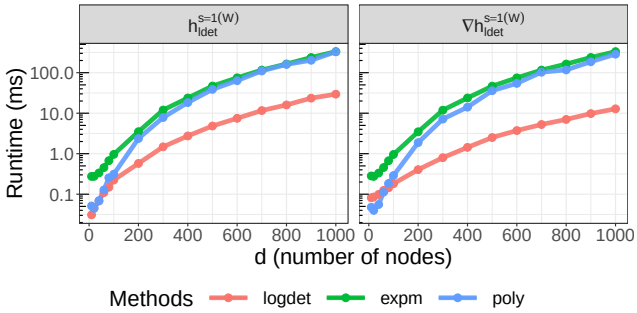


Figure 3: For each d, 30 matrices were randomly sampled from a standard Gaussian distribution.

[3]Note that here the sign of an edge is not important since $W \circ W$ will have all edge weights equal to $+1$.

4 Optimization

In the previous section we argued why the log-det function should be preferred in practice. Let f_θ denote a model with parameters θ for the functions f_j in (1), e.g., neural networks as in [58]. In this section we turn to the problem of minimizing a given score function $Q(f_\theta; \boldsymbol{X})$ constrained to $h^s_{\text{ldet}}(W(\theta)) = 0$. That is, we aim to solve:

$$\min_\theta Q(f_\theta; \boldsymbol{X}) + \beta_1 \|\theta\|_1 \quad \text{subject to} \;\; h^s_{\text{ldet}}(W(\theta)) = 0, \tag{6}$$

where we include the ℓ_1 regularizer to promote sparse solutions.

Since the inception of the purely continuous framework for learning DAGs in [57], almost all follow-up work uses the augmented Lagrangian (AML) scheme to tackle problem (6), and L-BFGS-B [38] for solving the sequence of unconstrained problems. Motivated by the properties of h^s_{ldet} given in 3.1, we propose a simpler scheme named DAGMA, based on solving a sequence of unconstrained problems in which h^s_{ldet} is simply seen as a regularizer. DAGMA resembles the central path approach of barrier methods [8, 38], or the classical path-following approach for solving lasso problems [e.g. 14]. Our method is given in Algorithm 1.

Algorithm 1 DAGMA

Require: Data matrix $\boldsymbol{X}$, initial central path coefficient $\mu^{(0)}$ (e.g., 1), decay factor $\alpha \in (0, 1)$ (e.g., 0.1), ℓ_1 parameter $\beta_1 > 0$ (e.g., 0.01), log-det parameter $s > 0$ (e.g., 1), number of iterations T.

1: Initialize $\theta^{(0)}$ so that $W(\theta^{(0)}) \in \mathbb{W}^s$.
2: **for** $t = 0, 1, 2, \ldots T-1$ **do**
3: Starting at $\theta^{(t)}$, solve $\theta^{(t+1)} = \arg\min_\theta \mu^{(t)}(Q(f_\theta; \boldsymbol{X}) + \beta_1 \|\theta\|_1) + h^s_{\text{ldet}}(W(\theta))$
4: Set $\mu^{(t+1)} = \alpha \mu^{(t)}$

Ensure: $W(\theta^{(T)})$

The following lemma states that DAGMA will return a DAG at the limit of the central path. This is a critical distinction against existing methods, many of which rely on some type of post-processing (e.g. thresholding) to ensure that the solution is a DAG.

Lemma 6. *Algorithm 1 is guaranteed to return a DAG whenever* $\mu^{(t)} \to 0$.

4.1 Practical Considerations

1. As in barrier methods, where it is required to start at the interior of the feasibility region, in Algorithm 1, we require that the initial point $W(\theta^{(0)})$ be inside $\mathbb{W}^s$. This is very easy to achieve since the zero matrix is in the interior of $\mathbb{W}^s$ for any $s > 0$; therefore, in our experiments we simply set $\theta^{(0)} = 0$.

2. Note that in Algorithm 1, we let $\mu^{(t)}$ decrease by a constant factor at each iteration; however, it is possible to specify explicitly the value of each $\mu^{(t)}$, e.g., for $T = 4$, we can let $\mu = \{1, 0.1, 0.001, 0\}$.

3. Regarding the choice of s, in principle s could take any value greater than zero since DAGs are inside $\mathbb{W}^s$ for any $s > 0$. Similar to μ, it is also possible to let s vary at each iteration, e.g., for $T = 4$, we can set $s = \{1, 0.9, 0.8, 0.8\}$. In practive, we observe that *slightly* decreasing s can help to obtain larger gradients as W gets closer to a DAG. Note, however, that letting s be equal or close to 1 is generally easier to optimize than setting s closer to zero, the reason being that for smaller values of s the volume of $\mathbb{W}^s$ is smaller and will require much smaller learning rates to stay inside $\mathbb{W}^s$, hence affecting convergence.

4. Finally, we do not specify how to solve line 3 in Algorithm 1, this is because we leave the door open for different solvers to be used. For our experiments in the next section, we solve line 3 by using a first-order method with the ADAM optimizer [24], which works remarkably well as shown in our experiments. It remains as future work to exploit the Hessian structure of h_{ldet} given in Lemma 3 for second-order methods.

5 Experiments

We compare our method against GES [11], PC [50], NOTEARS [57], and GOLEM [37] on both linear and nonlinear SEM. The source codes for each of the aforementioned methods are specified in Appendix C. Consistent with previous work in this area (e.g., NOTEARS and follow up work), we have not performed any hyperparameter optimization: This is to avoid presenting unintentionally biased results. As a concrete example, for each of the SEM settings, we simply chose a reasonable value for the ℓ_1 penalty coefficient and used that same value for all ER and SF graphs across many different numbers of nodes. Our experimental setting is similar to [57, 58]. Moreover, since the accuracy of certain methods were significantly lower than other methods, we only report the result against competitive methods in the main text; full results for all settings and methods can be found in Appendix C.

In the following figures, ER4 and SF4 denote the Erdős-Rényi and scale-free graphs, where, for each number of nodes d, each simulated graph has in expectation $4d$ number of edges. It is worth noting that the experimental settings by Zheng et al. [57, 58] consider graph models such as ER1, ER2, ER4, SF1, SF2, and SF4. Here we mainly focus in the **hardest** settings, that is, **ER4** and **SF4**. For the linear SEM setting, we present results for graphs with small to moderate number of nodes in Figure 4, and for graphs with large number of nodes in Figure 5. In both regimes, we observe that DAGMA obtains significant speedups and improvements in terms of structural accuracy against NOTEARS and GOLEM, even though GOLEM is *specific* to and specialized for linear Gaussian SEMs. For the nonlinear setting in Figure 6, in the generalized linear model (logistic), we similarly observe that DAGMA attains major speedups and improvements on SHD against NOTEARS [57]. For the general nonlinear model (neural networks), we observe that DAGMA is comparable in SHD to the NONLINEAR NOTEARS [58] but obtains about significant speedups. We invite the reader to look at Appendix C for more details and additional experiments.

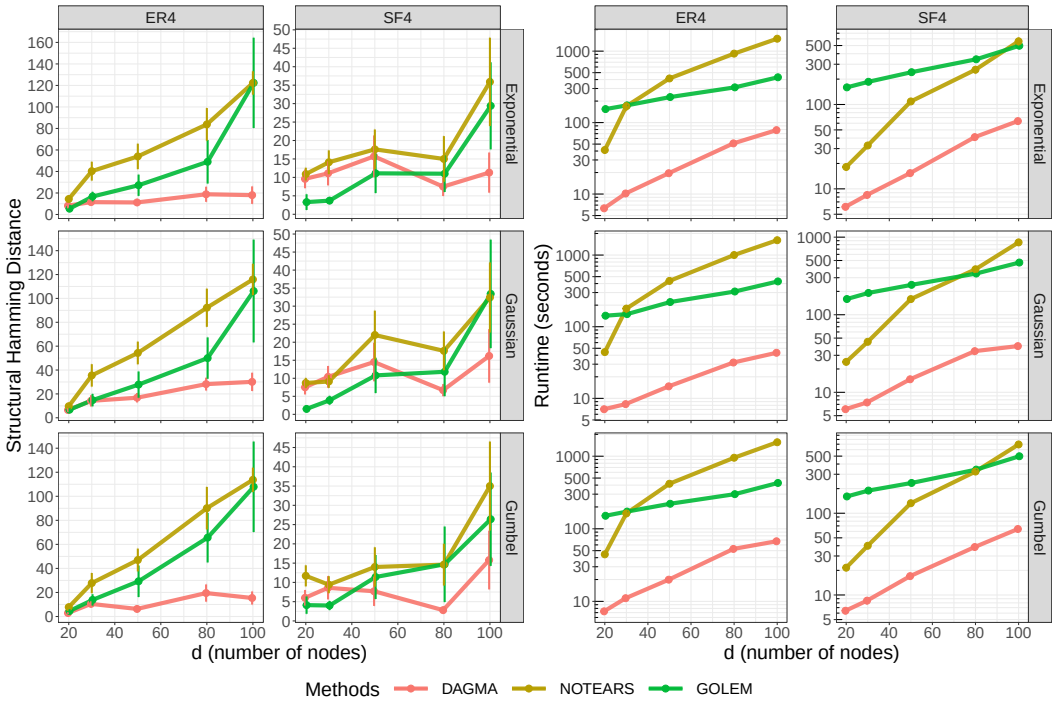


Figure 4: Experiments on linear SEMs for $d \in [20, 100]$. Each point in the plot is estimated over 10 repetitions, where error bars are the standard error. Wall time limit was set to 36 hours.

6 Final Remarks

A relevant assumption in this work is that of sufficiency, that is, there are no hidden variables that are a common cause of at least two observed variables. While this assumption is widely used for structure learning, we nonetheless highlight that in practice it is very difficult to find scenarios where

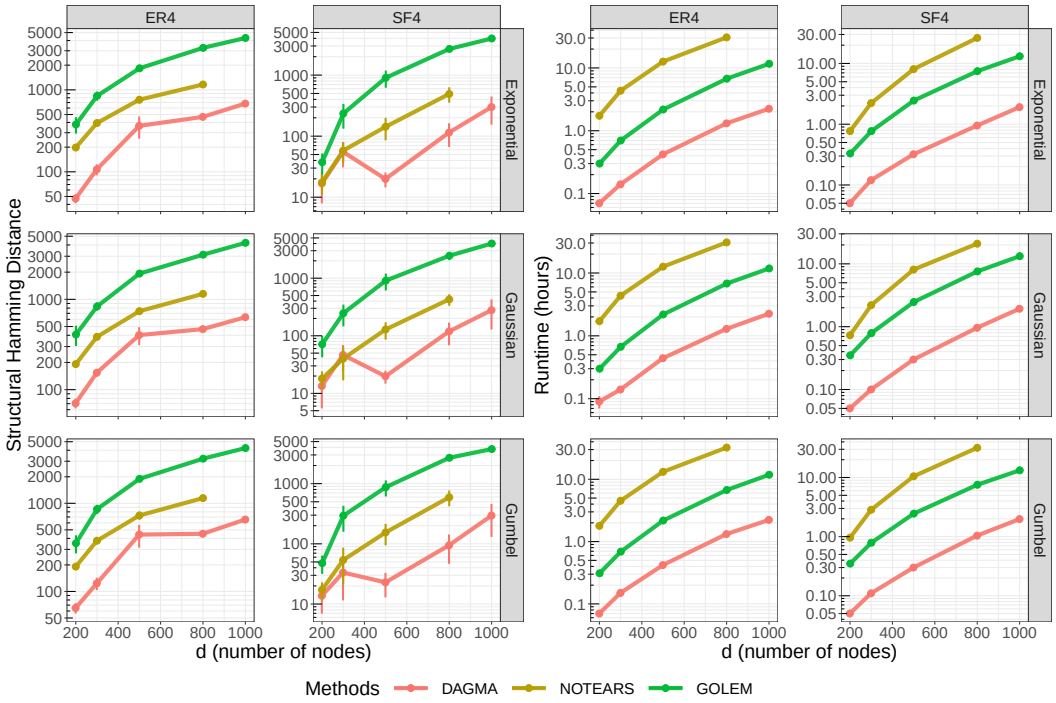


Figure 5: Experiments on linear SEMs for $d \in [200, 1000]$. Each point in the plot is estimated over 10 repetitions, where error bars are the standard error. Wall time limit was set to 36 hours.

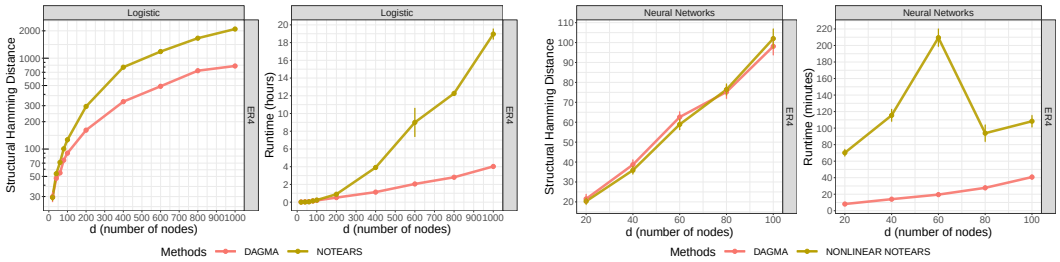


Figure 6: Experiments on nonlinear SEMs. Each point in the plot is estimated over 10 repetitions, where error bars are the standard error. Wall time limit was set to 36 hours.

such assumption holds. As with all work that assumes sufficiency, our work is an important necessary step to understanding settings with hidden variables. Finally, we note that the work by [6] proposes a differentiable approach for ADMG for the semi-Markovian case using h_{expm}. It is left for future work to explore the performance of such method using h_{ldet}.

Another important limitation of this and previous work on the continuous framework for learning DAGs is that of providing guarantees on the learned structure. As in real-life applications one does not have access to the ground-truth DAG, there is much uncertainty as to whether an edge in the predicted DAG actually corresponds to a causal relation. Thus, there is still a need for formal guarantees under the continuous framework.

Acknowledgments and Disclosure of Funding

K. B. was supported by NSF under Grant # 2127309 to the Computing Research Association for the CIFellows 2021 Project. B.A. was supported by NSF IIS-1956330, NIH R01GM140467, and the Robert H. Topel Faculty Research Fund at the University of Chicago Booth School of Business. P.R. was supported by ONR via N000141812861, and NSF via IIS-1909816, IIS-1955532, IIS-2211907. We are also grateful for the support of the University of Chicago Research Computing Center for assistance with the calculations carried out in this work.

References

[1] Al-Mohy, A. H. and Higham, N. J. [2011], 'Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators', *SIAM Journal on Scientific Computing* **33**(2), 488–511.

[2] Aragam, B., Amini, A. A. and Zhou, Q. [2019], 'Globally optimal score-based learning of directed acyclic graphs in high-dimensions', *Advances in Neural Information Processing Systems* .

[3] Aragam, B. and Zhou, Q. [2015], 'Concave Penalized Estimation of Sparse GaussianBayesian Networks', *Journal of Machine Learning Research* .

[4] Barabási, A.-L. and Albert, R. [1999], 'Emergence of scaling in random networks', *science* **286**(5439), 509–512.

[5] Berman, A. and Plemmons, R. J. [1994], *Nonnegative Matrices in the Mathematical Sciences*, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics.

[6] Bhattacharya, R., Nagarajan, T., Malinsky, D. and Shpitser, I. [2020], 'Differentiable Causal Discovery Under Unmeasured Confounding', *arXiv* .

[7] Bouckaert, R. R. [1993], Probabilistic network construction using the minimum description length principle, *in* 'European conference on symbolic and quantitative approaches to reasoning and uncertainty', Springer, pp. 41–48.

[8] Boyd, S. and Vandenberghe, L. [2004], *Convex Optimization*, Cambridge University Press.

[9] Bühlmann, P., Peters, J. and Ernest, J. [2014], 'Cam: Causal additive models, high-dimensional order search and penalized regression', *The Annals of Statistics* **42**(6), 2526–2556.

[10] Chickering, D. M. [1996], Learning bayesian networks is np-complete, *in* 'Learning from data', Springer, pp. 121–130.

[11] Chickering, D. M. [2003], 'Optimal structure identification with greedy search', *JMLR* **3**, 507–554.

[12] Chickering, D. M., Heckerman, D. and Meek, C. [2004], 'Large-sample learning of Bayesian networks is NP-hard', *Journal of Machine Learning Research* **5**, 1287–1330.

[13] Ernest, J., Rothenhäusler, D. and Bühlmann, P. [2016], 'Causal inference in partially linear structural equation models: identifiability and estimation', *arXiv preprint arXiv:1607.05980* .

[14] Friedman, J., Hastie, T., Höfling, H. and Tibshirani, R. [2007], 'Pathwise coordinate optimization', *The annals of applied statistics* **1**(2), 302–332.

[15] Gámez, J. A., Mateo, J. L. and Puerta, J. M. [2011], 'Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood', *Data Mining and Knowledge Discovery* **22**(1), 106–148.

[16] Ghoshal, A. and Honorio, J. [2017], Learning identifiable gaussian bayesian networks in polynomial time and sample complexity, *in* 'Proceedings of the 31st International Conference on Neural Information Processing Systems', pp. 6460–6469.

[17] Ghoshal, A. and Honorio, J. [2018], Learning linear structural equation models in polynomial time and sample complexity, *in* 'Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics', Vol. 84 of *Proceedings of Machine Learning Research*, PMLR, pp. 1466–1475.

[18] Goudet, O., Kalainathan, D., Caillou, P., Guyon, I., Lopez-Paz, D. and Sebag, M. [2018], Learning functional causal models with generative neural networks, *in* 'Explainable and interpretable models in computer vision and machine learning', Springer, pp. 39–80.

[19] Gu, J., Fu, F. and Zhou, Q. [2019], 'Penalized estimation of directed acyclic graphs from discrete data', *Statistics and Computing* **29**(1), 161–176.

[20] Hanson, M. A. [1981], 'On sufficiency of the Kuhn-Tucker conditions', *Journal of Mathematical Analysis and Applications* **80**(2), 545–550.

[21] 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.

[22] Higham, N. J. [2005], 'The scaling and squaring method for the matrix exponential revisited', *SIAM Journal on Matrix Analysis and Applications* **26**(4), 1179–1193.

[23] Hoyer, P., Janzing, D., Mooij, J. M., Peters, J. and Schölkopf, B. [2008], 'Nonlinear causal discovery with additive noise models', *Advances in neural information processing systems* **21**.

[24] Kingma, D. P. and Ba, J. [2014], 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

[25] Kyono, T., Zhang, Y. and van der Schaar, M. [2020], 'Castle: Regularization via auxiliary causal graph discovery', *Advances in Neural Information Processing Systems* **33**, 1501–1512.

[26] Lachapelle, S., Brouillard, P., Deleu, T. and Lacoste-Julien, S. [2020], Gradient-based neural dag learning, *in* 'International Conference on Learning Representations'.

[27] Loh, P.-L. and Buhlmann, P. [2014], 'High-Dimensional Learning of Linear Causal Networks via Inverse Covariance Estimation', *Journal of Machine Learning Research* .

[28] Magnus, J. R. and Neudecker, H. [1985], 'Matrix differential calculus with applications to simple, hadamard, and kronecker products', *Journal of Mathematical Psychology* **29**(4), 474–492.

[29] Margaritis, D. and Thrun, S. [1999], Bayesian network induction via local neighborhoods, *in* 'Proceedings of the 12th International Conference on Neural Information Processing Systems', pp. 505–511.

[30] Martin, D. H. [1985], 'The essence of invexity', *Journal of Optimization Theory and Applications* **47**(1), 65–76.

[31] Maxwell Chickering, D. and Heckerman, D. [1997], 'Efficient approximations for the marginal likelihood of bayesian networks with hidden variables', *Machine learning* **29**(2), 181–212.

[32] Meinshausen, N. and Bühlmann, P. [2006], 'High-dimensional graphs and variable selection with the Lasso', *The Annals of Statistics* **34**(3).

[33] Moler, C. and Van Loan, C. [2003], 'Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later', *SIAM review* **45**(1), 3–49.

[34] Monti, R. P., Zhang, K. and Hyvärinen, A. [2020], Causal discovery with general non-linear relationships using non-linear ica, *in* 'Uncertainty in Artificial Intelligence', PMLR, pp. 186–195.

[35] Mooij, J. M., Peters, J., Janzing, D., Zscheischler, J. and Schölkopf, B. [2016], 'Distinguishing cause from effect using observational data: methods and benchmarks', *The Journal of Machine Learning Research* **17**(1), 1103–1204.

[36] Moraffah, R., Moraffah, B., Karami, M., Raglin, A. and Liu, H. [2020], 'Causal adversarial network for learning conditional and interventional distributions', *arXiv preprint arXiv:2008.11376* .

[37] Ng, I., Ghassami, A. and Zhang, K. [2020], 'On the Role of Sparsity and DAG Constraints for Learning Linear DAGs', *arXiv* .

[38] Nocedal, J. and Wright, S. J. [2006], *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering.

[39] Ostrowski, A. [1937], 'Über die determinanten mit überwiegender Hauptdiagonale', *Commentarii Mathematici Helvetici* **10**(1), 69–96.

[40] Pamfil, R., Sriwattanaworachai, N., Desai, S., Pilgerstorfer, P., Georgatzis, K., Beaumont, P. and Aragam, B. [2020], Dynotears: Structure learning from time-series data, *in* 'International Conference on Artificial Intelligence and Statistics', PMLR, pp. 1595–1605.

[41] Park, G. and Park, S. [2019], 'High-dimensional poisson structural equation model learning via \ell_1-regularized regression.', *J. Mach. Learn. Res.* **20**, 95–1.

[42] Park, G. and Raskutti, G. [2017], 'Learning quadratic variance function (qvf) dag models via overdispersion scoring (ods).', *J. Mach. Learn. Res.* **18**, 224–1.

[43] Pearl, J. [2009], *CAUSALITY: Models, Reasoning, and Inference*, 2nd edn, Cambridge University Press.

[44] Peters, J. and Bühlmann, P. [2014], 'Identifiability of gaussian structural equation models with equal error variances', *Biometrika* **101**(1), 219–228.

[45] Peters, J., Mooij, J. M., Janzing, D. and Bernhard [2014], 'Causal Discovery with Continuous Additive Noise Models', *Journal of Machine Learning Research* .

[46] Petersen, K. B. and Pedersen, M. S. [2012], *The Matrix Cookbook*.

[47] Ramsey, J., Glymour, M., Sanchez-Romero, R. and Glymour, C. [2017], 'A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images', *International journal of data science and analytics* **3**(2), 121–129.

[48] Shimizu, S., Hoyer, P. O., Hyvärinen, A., Kerminen, A. and Jordan, M. [2006], 'A linear non-gaussian acyclic model for causal discovery.', *Journal of Machine Learning Research* **7**(10).

[49] Spirtes, P. and Glymour, C. [1991], 'An algorithm for fast recovery of sparse causal graphs', *Social Science Computer Review* **9**(1), 62–72.

[50] Spirtes, P., Glymour, C. and Scheines, R. [2000], *Causation, Prediction, and Search*, The MIT Press.

[51] Tsamardinos, I., Aliferis, C. F., Statnikov, A. R. and Statnikov, E. [2003], Algorithms for large scale markov blanket discovery, *in* 'FLAIRS conference', Vol. 2, pp. 376–380.

[52] Tsamardinos, I., Brown, L. E. and Aliferis, C. F. [2006], 'The max-min hill-climbing Bayesian network structure learning algorithm', *Machine Learning* **65**(1), 31–78.

[53] Voorman, A., Shojaie, A. and Witten, D. [2014], 'Graph estimation with joint additive models', *Biometrika* **101**(1), 85–101.

[54] Wei, D., Gao, T. and Yu, Y. [2020], 'DAGs with No Fears: A Closer Look at Continuous Optimization for Learning Bayesian Networks', *arXiv* .

[55] Yu, Y., Chen, J., Gao, T. and Yu, M. [2019], 'DAG-GNN: DAG Structure Learning with Graph Neural Networks', *arXiv* .

[56] Yu, Y., Gao, T., Yin, N. and Ji, Q. [2021], 'DAGs with No Curl: An Efficient DAG Structure Learning Approach', *arXiv* .

[57] Zheng, X., Aragam, B., Ravikumar, P. and Xing, E. P. [2018], 'DAGs with NO TEARS: Continuous Optimization for Structure Learning', *arXiv* .

[58] Zheng, X., Dan, C., Aragam, B., Ravikumar, P. and Xing, E. P. [2020], 'Learning Sparse Nonparametric DAGs', *International Conference on Artificial Intelligence and Statistics* .

[59] Zhu, S., Ng, I. and Chen, Z. [2020], Causal discovery with reinforcement learning, *in* 'International Conference on Learning Representations'.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] (See Section 6)
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] (See Section D)
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] (See Section 2 and 3)
   (b) Did you include complete proofs of all theoretical results? [Yes] (See Section A)
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] (See Section C)
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] (See Section C)

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] (See Section C)

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] (See Section C)

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes] (See Section C)

   (b) Did you mention the license of the assets? [N/A]

   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

SUPPLEMENTARY MATERIAL
DAGMA: Learning DAGs via M-matrices and a Log-Determinant Acyclicity Characterization

A Detailed Proofs

A.1 Proof of Theorem 1

Proof. We first note that for any $s > 0$ and matrix $A \in \mathbb{R}^{d\times d}$ we have $\det(sA) = s^d A$. Then, $\log\det(sI - W\circ W) - d\log s = \log(s^d \det(I - s^{-1}W\circ W)) - d\log s = \log\det(I - s^{-1}W\circ W)$. Moreover, since $W \in \mathbb{W}^s$, we have that $s > \rho(W\circ W)$ or equivalently $1 > \rho(s^{-1}W\circ W)$. Thus, in the sequel of the proof we set $s = 1$ w.l.o.g.

Item (ii). The gradient expression follows from standard matrix calculus [46]. From Lemma 2, it follows that W is an stationary point of $h^{s=1}_{\text{ldet}}(W)$, i.e., $\nabla h^{s=1}_{\text{ldet}}(W) = 0$, if and only if W corresponds to a DAG.

Item (i). From the item above, we characterized the stationary points of $h^{s=1}_{\text{ldet}}$. Moreover, from Proposition 1, we know that for any $W \in \mathbb{W}^s$, the gradient $\nabla h^{s=1}_{\text{ldet}}$ is well-defined since $I - W\circ W$ is an M-matrix and, thus, its inverse exists. Finally, note that at the boundary of $\mathbb{W}^s$, we have that $h^{s=1}_{\text{ldet}}(W) \to \infty$. From these observations, we have that the global minima of $h^{s=1}_{\text{ldet}}$ must be in the interior of $\mathbb{W}^s$ and will correspond to the set of stationary points. Hence, DAGs are local and global minima of $h^{s=1}_{\text{ldet}}$.

We conclude by noting that if W is a DAG then we have $\det(I - W\circ W) = 1$ and the equality $h^{s=1}_{\text{ldet}}(W) = 0$ holds immediately. Since DAGs are global minima, this implies that for all $W \in \mathbb{W}^s$ we have $h^{s=1}_{\text{ldet}}(W) \geq 0$. □

A.2 Proof of Lemma 1

Proof. **Item (i).** The proof follows directly from the fact that the weighted adjacency matrix W of any DAG is a nilpotent matrix. Since $W\circ W$ is also a nilpotent matrix, its spectral radius is zero, i.e., $\rho(W\circ W) = 0$. Thus, for any $s > 0$, if W is a DAG then $W \in \mathbb{W}^s$.

Item (ii). Recall that a space $\mathcal{X}$ is path-connected if, for any two points $x, y \in \mathcal{X}$, there exists a continuous function (path) $\phi : [0,1] \to \mathcal{X}$ such that $\phi(0) = x$ and $\phi(1) = y$. Note that since the $d\times d$ zero matrix, $\mathbf{0}$, has spectral radius zero, it clearly follows that $\mathbf{0} \in \mathbb{W}^s$ for any $s > 0$. We prove path-connectedness of $\mathbb{W}^s$ by showing that for any $W \in \mathbb{W}^s$ there exist a path ϕ to $\mathbf{0}$. Then, for any $W \in \mathbb{W}^s$, define $\phi(t) = (1-t)W$. It is clear that ϕ is a continuous function on t, where $\phi(0) = W \in \mathbb{W}^s$ and $\phi(1) = 0 \in \mathbb{W}^s$. Now we need to show that $\phi(t) \in \mathbb{W}^s$ for all $t \in (0,1)$. Let t_1 be an arbitrary number in $(0,1)$, by the nonnegativity of $W\circ W$ and by $(1-t_1)^2 < 1$, we have that $(1-t_1)^2 W\circ W < W\circ W$. Finally, by Perron-Frobenius theory on nonnegative matrices, we have that $\rho\left((1-t_1)^2 W\circ W\right) < \rho\left(W\circ W\right) < s$, where the last inequality follows by $W \in \mathbb{W}^s$, which implies that $\phi(t_1) \in \mathbb{W}^s$. As the choice of t_1 was arbitrary, we conclude the proof.

Item (iii). The proof follows immediately by the definition of $\mathbb{W}^s$. □

A.3 Proof of Lemma 2

Proof. First, recall that $\nabla h^s_{\text{ldet}}(W) = 2(sI - W\circ W)^{-\top}\circ W$. Second, since $(sI - W\circ W)$ is an M-matrix, by Proposition 1, we have that $(sI - W\circ W)^{-\top} \geq 0$. By the latter, whenever $[\nabla h^s_{\text{ldet}}(W)]_{i,j} \neq 0$, we have that $\text{sign}([\nabla h^s_{\text{ldet}}(W)]_{i,j}) = \text{sign}(W_{i,j})$. Finally, from the series expansion of the inverse we have:

$$(sI - W\circ W)^{-1} = \frac{1}{s}I + \frac{1}{s^2}(W\circ W) + \frac{1}{s^3}(W\circ W)^2 + \cdots,$$

by taking the transpose, that implies that the i, j entry $[(sI - W \circ W)^{-\top}]_{i,j}$ is nonzero if and only if there exists a directed walk from j to i. By taking the Hadamard product, we have that $[(sI - W \circ W)^{-\top} \circ W]_{i,j}$ is nonzero if and only if $W_{i,j} \neq 0$ and $[(sI - W \circ W)^{-\top}]_{i,j} \neq 0$, i.e., there must exist a closed walk of the form $i \to j \rightsquigarrow i$. Which concludes the proof. □

A.4 Proof of Lemma 3

Proof. We use the Magnus-Neudecker convention [28] for calculating the Hessian. Then, by taking differentials and vectorizing, we obtain:

$$\begin{aligned}
\partial(\nabla h(W)) &= 2\, \partial N^\top \circ W + 2\, N^\top \circ \partial W \\
\operatorname{vec}(\partial(\nabla h(W))) &= 2\, \operatorname{vec}(\partial N^\top \circ W) + 2\, \operatorname{vec}(N^\top \circ \partial W) \\
&= 2\, \operatorname{Diag}(\operatorname{vec}(W))\operatorname{vec}(\partial N^\top) + 2\, \operatorname{Diag}(\operatorname{vec}(N^\top))\operatorname{vec}(\partial W). \qquad (7)
\end{aligned}$$

Recall that $N = (sI - W \circ W)^{-1}$, we now derive the expression for $\operatorname{vec}(\partial N^\top)$,

$$\begin{aligned}
\operatorname{vec}(\partial N^\top) &= -(N \otimes N^\top)\, \operatorname{vec}(\partial(sI - W \circ W)^\top) \\
&= 2\, (N \otimes N^\top)\, \operatorname{vec}(W^\top \circ \partial W^\top) \\
&= 2\, (N \otimes N^\top)\, \operatorname{Diag}(\operatorname{vec}(W^\top))\operatorname{vec}(\partial W^\top) \\
&= 2\, (N \otimes N^\top)\, \operatorname{Diag}(\operatorname{vec}(W^\top))K^{dd}\, \operatorname{vec}(\partial W),
\end{aligned}$$

plugging in the last equality into eq.(7), we have

$$\begin{aligned}
\nabla^2 h(W) &= \frac{\partial\, \operatorname{vec}(\nabla h(W))}{\partial\, \operatorname{vec}(W)} \\
&= 4\, \operatorname{Diag}(\operatorname{vec}(W))(N \otimes N^\top)\, \operatorname{Diag}(\operatorname{vec}(W^\top))K^{dd} + 2\, \operatorname{Diag}(\operatorname{vec}(N^\top)),
\end{aligned}$$

which concludes the proof. □

A.5 Proof of Lemma 4

Proof. The comparison between h_{expm} and h_{poly} is straightforward by looking at the coefficients of their series expansions. Recall that, $h_{\text{expm}}(W) = \sum_{k=0}^{\infty} {}^{1}\!/_{k!}\, \operatorname{Tr}((W \circ W)^k) - d$ and $h_{\text{poly}}(W) = \sum_{k=0}^{d} \binom{d}{k}/d^k\, \operatorname{Tr}((W \circ W)^k) - d$. Since ${}^{1}\!/_{k!} \geq \binom{d}{k}/d^k$, it is clear that $h_{\text{expm}}(W) \geq h_{\text{poly}}(W)$. To prove that $h_{\text{ldet}}^{s=1}(W) \geq h_{\text{expm}}(W)$, we use the fact that every square matrix has a Jordan canonical form. Let $W \circ W = Q^{-1}JQ$, where Q is an invertible matrix and J is in Jordan normal form (i.e., a block diagonal matrix with 1s in the super-diagonal), we have that $\operatorname{Tr}(e^{W \circ W}) = \operatorname{Tr}(e^J)$. Let $\Lambda(W \circ W)$ be the set of distinct generalized eigenvalues of $W \circ W$, and m_λ be the multiplicity corresponding to $\lambda \in \Lambda(W \circ W)$. Then, we have that

$$h_{\text{expm}}(W) = \operatorname{Tr}(e^{W \circ W}) - d = \sum_{\lambda \in \Lambda(W \circ W)} m_\lambda (e^\lambda - 1).$$

From $h_{\text{ldet}}^{s=1}(W)$ we have,

$$h_{\text{ldet}}^{s=1}(W) = -\log\det(I - W \circ W) = \sum_{\lambda \in \Lambda(W \circ W)} m_\lambda (-\log(1 - \lambda)),$$

where $\log$ denotes the principal branch of the complex logarithm. For any complex λ, we have the Taylor series:

$$e^\lambda - 1 = \sum_{k=1}^{\infty} \frac{\lambda^k}{k!}, \qquad -\log(1 - \lambda) = \sum_{k=1}^{\infty} \frac{\lambda^k}{k},$$

where both series converges precisely for all complex numbers $|\lambda| \leq 1, \lambda \neq 1$, which is the case as $W \in \mathbb{W}^{s=1}$. From the latter, it is clear to see that $h_{\text{expm}}(W) \leq h_{\text{ldet}}^{s=1}(W)$, which conludes the proof. □

A.6 Proof of Lemma 5

Proof. First let us write the gradients for the different acyclicity characterizations. Then, we have $\nabla h_{\text{expm}}(W) = 2(e^{W \circ W})^\top \circ W$, $\nabla h_{\text{poly}}(W) = 2((I + \frac{1}{d} W \circ W)^{d-1})^\top \circ W$, and $\nabla h^{s=1}_{\text{ldet}}(W) = 2((I - W \circ W)^{-1})^\top \circ W$. When taking absolute values, it is clear that they differ due to the left-hand side of each Hadamard product. Thus, we need to look at the entries of: $|e^{W \circ W}|$, $|(I + \frac{1}{d} W \circ W)^{d-1}|$, and $|(I - W \circ W)^{-1}|$. From their series expansions we have:

$$
\begin{aligned}
e^{W \circ W} &= \sum_{k=0}^{\infty} \frac{1}{k!} (W \circ W)^k, \\
(I + \frac{1}{d} W \circ W)^{d-1} &= \sum_{k=0}^{d-1} \frac{\binom{d-1}{k}}{(d-1)^k} (W \circ W)^k, \\
(I - W \circ W)^{-1} &= \sum_{k=0}^{\infty} (W \circ W)^k.
\end{aligned}
$$

Since $W \circ W$ is nonnegative, each power $(W \circ W)^k$ is also nonnegative. Therefore, by noting that $\binom{d-1}{k} / (d-1)^k \leq 1/k! \leq 1$ for all k, the statement follows. □

A.7 Proof of Lemma 6

Proof. The proof follows by noting that at the limit of the central path ($\mu^{(t)} \to 0$) we solve the following problem:

$$
\widehat{\theta} = \arg\min_{\theta} h^s_{\text{ldet}}(W(\theta)).
$$

Then, by the invexity property of h^s_{ldet} (see Corollary 3), it follows that the solution $W(\widehat{\theta})$ must be a DAG. □

A.8 Proof of Corollary 1

Proof. For any $W \in \mathbb{W}^s$, from Lemma 2, we know that the nonzeros of $\nabla h^s_{\text{ldet}}(W)$ have the same sign as the corresponding entries in W. Then, let $Y = W - a \nabla h^s_{\text{ldet}}(W)$ for a small value a such that $|Y| \leq |W|$. It follows that $Y \circ Y \leq W \circ W$. Since $Y \circ Y$ and $W \circ W$ are nonnegative matrices, by Perron-Frobenius, we have that $\rho(Y \circ Y) \leq \rho(W \circ W)$. The latter implies that $Y \in \mathbb{W}^s$, thus, the negative gradient, $-\nabla h^s_{\text{ldet}}$, must point towards the interior of $\mathbb{W}^s$. □

A.9 Proof of Corollary 2

Proof. Given the Hessian expression in Lemma 3, the expressions for its entries follow by simple algebraic manipulation. From the argument in the proof of Lemma 2 (see Apeendix A.3), we have that $N_{i,j} > 0$ if and only if there exist a directed walk from i to j. By the latter, the signs and cycle interpretations follow. □

A.10 Proof of Corollary 3

Proof. Follows from Theorem 1. □

B Additional Discussions

B.1 Additional Example for Section 3.1

In Figure 1, we provided an example of a two-node graph to visualize the properties of h_{ldet}. In Figure 7, we present another example for a three-node graph with three edges (parameters). Specifically, the graph is parameterized by $W = \begin{bmatrix} 0 & w_1 & 0 \\ 0 & 0 & w_2 \\ 0 & w_3 & 0 \end{bmatrix}$. Here note that for W to be a DAG at least one of w_2 or w_3 must be zero.

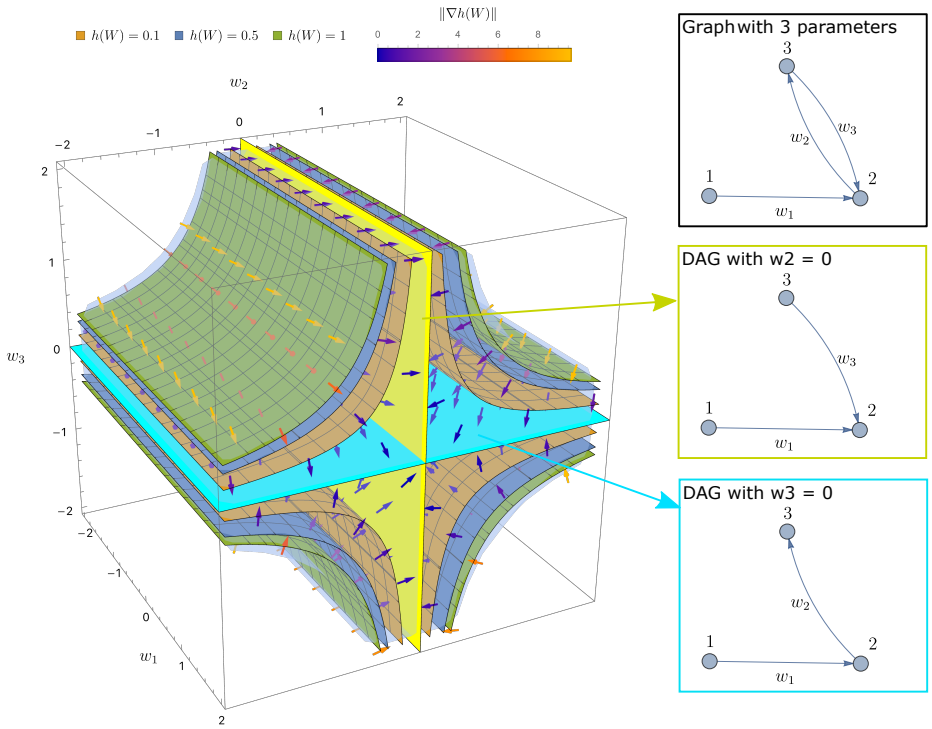


Figure 7: The curved manifolds represent the level sets of $h^{s=1}_{\text{ldet}}$. The arrows represent the vector field $\nabla h^{s=1}_{\text{ldet}}$. The yellow plane represents DAGs where $w_2 = 0$, while the cyan plane represents DAGs where $w_3 = 0$. Similar to Figure 1, we observe that the negative gradients point towards the interior of $\mathbb{W}^{s=1}$ and that DAGs are stationary points (attractors) of $h^{s=1}_{\text{ldet}}$.

B.2 Further Details for Section 3.2

In this section, we expand the discussion given in Section 3.2

Argument (i). h^s_{ldet} does not diminish cycles of any length. Let $h_{\text{tinv}}(W) = \text{Tr}((I - W \circ W)^{-1}) = \sum_{k=0}^{\infty} \text{Tr}((W \circ W)^k) - d$. We note that h_{tinv} is another acyclicity characterization, previously considered by Zheng et al. [57]. From its series expansion, the reader might wonder if simply considering $h_{\text{tinv}}(W)$ would be a better alternative than $h_{\text{ldet}}(W)$, $h_{\text{expm}}(W)$, and $h_{\text{poly}}(W)$. The answer is *no*: While this alternative characterization does not suffer from the dimishing cycle problem, it is prone to numerical instability, which was already noted in [57]. This numerical instability is mainly due to the exploding gradients of $\text{Tr}((I - W \circ W)^{-1})$, which we discuss in our next argument.

Argument (ii). h^s_{ldet} has better behaved gradients. Let us write the gradients of the different acyclicity characterizations. We have:

$$\nabla h_{\text{expm}}(W) = 2(e^{W \circ W})^\top \circ W, \qquad \nabla h_{\text{poly}}(W) = 2((I + \frac{1}{d} W \circ W)^{d-1})^\top \circ W,$$

$$\nabla h_{\text{tinv}}(W) = 2((I - W \circ W)^{-2})^\top \circ W, \qquad \nabla h^s_{\text{ldet}}(W) = 2((sI - W \circ W)^{-1})^\top \circ W.$$

From the series expansion for $(I - W \circ W)^{-2}$ we obtain:

$$(I - W \circ W)^{-2} = \sum_{k=0}^{\infty} (k+1)(W \circ W)^k.$$

Then, ∇h_{tinv} is numerically unstable whenever $W \circ W$ has walks of moderate weight since all walks of length k are now weighted by $k + 1$, thus, being prone to exploding gradients. Similar to argument (i), we have the following series expansions:

$$e^{W \circ W} = \sum_{k=0}^{\infty} \frac{1}{k!} (W \circ W)^k, \qquad (I + \frac{1}{d} W \circ W)^{d-1} = \sum_{k=0}^{d-1} \frac{\binom{d-1}{k}}{(d-1)^k} (W \circ W)^k,$$

and

$$(sI - W \circ W)^{-1} = \sum_{k=0}^{\infty} \frac{1}{s^{k+1}} (W \circ W)^k.$$

One can observe that for $s = 1$, the gradient of the log-determinant weights equally all cycles of any length, whereas ∇h_{expm} and ∇h_{poly} are again susceptible to the vanishing cycle problem and, thus, they might suffer from vanishing gradients. The plot in Figure 2 shows how the gradients of h_{expm} and h_{poly} decay at a very fast rate as the cycle graph has more nodes.

Argument (iii). Computing h^s_{ldet} and ∇h^s_{ldet} is empirically faster. In Figure 3, we compared the runtimes of h^s_{ldet}, h_{expm} and h_{poly} for randomly generated matrices. We used the benchmarking library from PyTorch for better runtime estimates over single threads. Experiments were conducted on an Intel Xeon processor E5 v4 with 2.40 GHz frequency.

The reason that computing h^s_{ldet} and ∇h^s_{ldet} is faster is that it involves computing a log-determinant and a matrix inverse respectively, and both of these problems enjoy the large body of work on optimized libraries for matrix factorizations (e.g., LU decomposition) and linear-system solvers. In contrast, computing h_{expm} relies on a truncated Taylor series of the exponential whose degree is typically estimated using scaling [22, 1], and this requires several matrix-matrix multiplications. The matrix exponential is also a notoriously tricky object to compute [33]. Similar to h_{expm}, computing h_{poly} also requires several matrix-matrix multiplications and, thus, both attain similar performances.

C Detailed Experiments

Computing. All experiments were conducted on an 8-core Intel Xeon processor E5-2680v4 with 2.40 GHz frequency, and 32GB of memory. Each experiment had a wall time of 36 hours.

Graph Models. Each simulation in our experiments samples a graph from two well-known random graph models:

- **Erdos-Rényi** (ER) graphs: These are random graphs whose edges are added independently with equal probability. We use the notation ERk to indicate that the graph model is an ER graph with kd edges in expectation.
- **Scale-free** (SF) graphs: These are random graphs simulated according to the preferential attachment process [4]. We use the notation SFk to indicate that the graph model is an SF graph with kd edges in expectation and $\beta = 1$, where β is the exponent used in the preferential attachment process. It is worth noting that since we consider directed graphs, this particular model corresponds to Price's model, a classical graph model for the growth of citation networks

Note that ER graphs are random *undirected* graphs. To produce a DAG, we draw a random permutation of numbers from 1 to d and orient the edges respecting this vertex ordering. For the case of SF graphs, the edges are oriented each time a new node is attached, thus, the sampled graph is a DAG. Once the ground-truth DAG is generated, we need to simulate the structural equation model, where we provide experiments for linear and nonlinear SEMs.

Remark 4. *It is worth noting that the experimental settings by Zheng et al. [57, 58] consider graph models such as ER1, ER2, ER4, SF1, SF2, and SF4. Here we mainly focus in the* ***hardest*** *settings, that is,* ***ER4*** *and* ***SF4****.*

Metrics. We evaluate the performance of each algorithm with the following four metrics:

- **Structure Hamming distance (SHD)**: A standard measurement for structure learning that counts the total number of edges additions, deletions, and reversals needed to convert the estimated graph into the true graph.
- **True Positive Rate (TPR)**: Measures the proportion of *correctly* identified edges with respect to the total number of edges in the ground-truth DAG.
- **False Positive Rate (FPR)**: Measures the proportion of *incorrectly* identified edges with respect to the total number of *absent* edges in the ground-truth DAG.

- **Runtime**: Measures how much time the algorithm takes to run, we use it to measure the speed of the algorithms.

Remark 5. *Consistent with previous work in this area (e.g. NOTEARS and its follow-ups), we have not performed any hyperparameter optimization: This is to avoid presenting unintentionally biased results. As a concrete example, for each of the following SEM settings, we simply chose a reasonable value for the ℓ_1 penalty coefficient and used that same value for all ER and SF graphs across many different numbers of nodes.*

C.1 SEM: Linear Setting

In the linear case, the functions f_j in (1) are directly parameterized by the weighted adjacency matrix W. That is, we have the following set of equations:

$$X_j = w_j^\top X + Z_j,$$

where $W = [w_1 \mid \cdots \mid w_d] \in \mathbb{R}^{d\times d}$, and $Z_j \in \mathbb{R}$ represents the noise. Here W encodes the graphical structure, i.e., there is an edge $X_k \to X_j$ if and only if $W_{k,j} \neq 0$.

Then, given the ground-truth DAG $B \in \{0,1\}^{d\times d}$ from one of the two graph models ER or SF, we assigned edge weights independently from $\mathrm{Unif}\left([-2,-0.5] \cup [0.5,2]\right)$ to obtain a weight matrix $W \in \mathbb{R}^{d\times d}$. Given W, we sampled $X = W^\top X + Z \in \mathbb{R}^d$ according to the following three noise models:

- **Gaussian noise:** $Z_j \sim \mathcal{N}(0,1), \forall j \in [d]$.
- **Exponential noise:** $Z_j \sim \mathrm{Exp}(1), \forall j \in [d]$.
- **Gumbel noise:** $Z_j \sim \mathrm{Gumbel}(0,1), \forall j \in [d]$.

Based on these models, we generated random datasets $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ by generating the rows i.i.d. according to one of the models above. For each simulation, we generated $n = 1000$ samples, unless otherwise stated.

To measure the quality of a model, we use the least-square loss

$$Q(W;\mathbf{X}) = \frac{1}{2n}\|\mathbf{X} - \mathbf{X}W\|_F^2, \tag{8}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. The coefficient β_1 used for ℓ_1 regularization, and other parameters required for DAGMA (see Algorithm 1), are later specified for each figure.

The implementation details of the baselines are listed below:

- GES (specifically, the FGES algorithm in [47]) and PC [50] are standard baselines for structure learning. Their implementation is based on the `py-causal` package, available at `https://github.com/bd2kccd/py-causal`. The exact set of hyperparameters used are:
  - For PC: `testId = 'fisher-z-test', depth = 3, fasRule = 2, dataType = 'continuous', conflictRule = 1, concurrentFAS = True, useMaxPOrientationHeuristic = True`.
  - For GES: `scoreId = 'cg-bic-score', maxDegree = 5, dataType = 'continuous', faithfulnessAssumed = False`.
- The NOTEARS method in Zheng et al. [57] was implemented using the author's Python code available at: `https://github.com/xunzheng/notears`. Its score function is also the least square as defined in eq.(8). For the ℓ_1 coefficient, for a fair comparison, we use the same value used for DAGMA. For the rest of hyperparameters, we use their default values.
- The GOLEM method in Ng et al. [37] was implemented using the author's Python and Tensorflow code available at: `https://github.com/ignavierng/golem`. Here we use their default set of hyperparameters, that is, $\lambda_1 = 0.02$ and $\lambda_2 = 5$, for other details of their method we refer the reader to Appendix F of Ng et al. [37].

C.1.1 Small to Moderate Number of Nodes

Following the aforementioned process to generate data, in this section, we test the methods for graphs with number of variables $d \in \{20, 30, 50, 80, 100\}$. We use the following setting for DAGMA (Algorithm 1): Number of iterations $T = 4$, initial central path coefficient $\mu^{(0)} = 1$, decay factor $\alpha = 0.1$, ℓ_1 coefficient $\beta_1 = 0.05$, log-det parameter $s = \{1, .9, .8, .7\}$. For each problem in line 3 of Algorithm 1, we implement an adaptive gradient method using the ADAM optimizer [24]. The hyperparameters for ADAM are: Learning rate of 3×10^{-4}, and $(\beta_1, \beta_2) = (0.99, 0.999)$. For $t = \{0, 1, 2\}$, we run ADAM for 2×10^4 iterations or until the loss converges, whichever comes first. For $t = 3$, we run ADAM for 7×10^4 iterations or until the loss converges, whichever comes first. We consider that the loss converges if the relative error between subsequent iterations is less than 10^{-6}. Finally, as in previous work including the baseline methods [57, 58, 37], a final thresholding step is performed as it was shown to help reduce the number of false discoveries. For all cases, we use a threshold of 0.3.

The results for different graph models (ER4, SF4), and different noise distributions (Gaussian, Gumbel, exponential), are shown in Figure 8. In Table 1, we average the SHDs and runtimes across graph and noise types, for the competitive methods. Here we note in particular that for $d = 100$, DAGMA obtains an **improvement** of 74.9% and 76.5% in **SHD** against GOLEM and NOTEARS, respectively; also, DAGMA runs 7.7 and 19.1 times **faster** than GOLEM and NOTEARS, respectively. Finally, we note that DAGMA performs much better than GOLEM besides the latter being *tailored* to linear Gaussian models.

Table 1: Summary of performances (SHD and runtime) of the most competitive methods. Each metric was averaged across different graph and noise types. Finally, the errors denote 95% confidence intervals on 10 repetitions.

Method	d	SHD	Runtime (seconds)
DAGMA	20	6.78±1.64	6.54± 0.42
	30	11.05±2.50	8.99± 0.62
	50	12.03±3.42	16.88±0.98
	80	13.92±4.44	41.55±3.30
	100	17.80±5.72	59.36±4.80
GOLEM	20	4.28±1.38	154.64±2.50
	30	9.48±3.10	177.21±5.62
	50	19.60±7.30	231.53±3.58
	80	33.68±12.87	324.41±7.10
	100	70.95±26.11	458.94±8.62
NOTEARS	20	10.53±1.58	32.41±3.72
	30	22.70±6.04	104.76±20.43
	50	34.82±7.96	278.13±40.14
	80	52.22±13.49	640.95±89.24
	100	75.87±13.97	1129.10±120.74

C.1.2 Large Number of Nodes

In this section, we test DAGMA, GOLEM, and NOTEARS for graphs with number of variables $d \in \{200, 300, 500, 800, 1000\}$. We do not test PC and GES as they are not competitive in terms of accuracy, as shown in Figure 8. We follow the same setting for DAGMA given in Section C.1.1.

The results for different graph models (ER4, SF4), and different noise distributions (Gaussian, Gumbel, exponential), are shown in Figure 9. In Table 2, we average the SHDs and runtimes across graph and noise types. Here we note in particular that for $d = 800$, DAGMA obtains an **improvement** of 90.2% and 65.5% in **SHD** against GOLEM and NOTEARS, respectively; also, DAGMA runs 6.2 and 25 times **faster** than GOLEM and NOTEARS, respectively. For $d = 1000$, we observe that NOTEARS takes more than 36 hours, which is the reason we could not report its performance. Finally, we note that once again DAGMA performs much better than GOLEM besides the latter being *tailored* to linear Gaussian models.

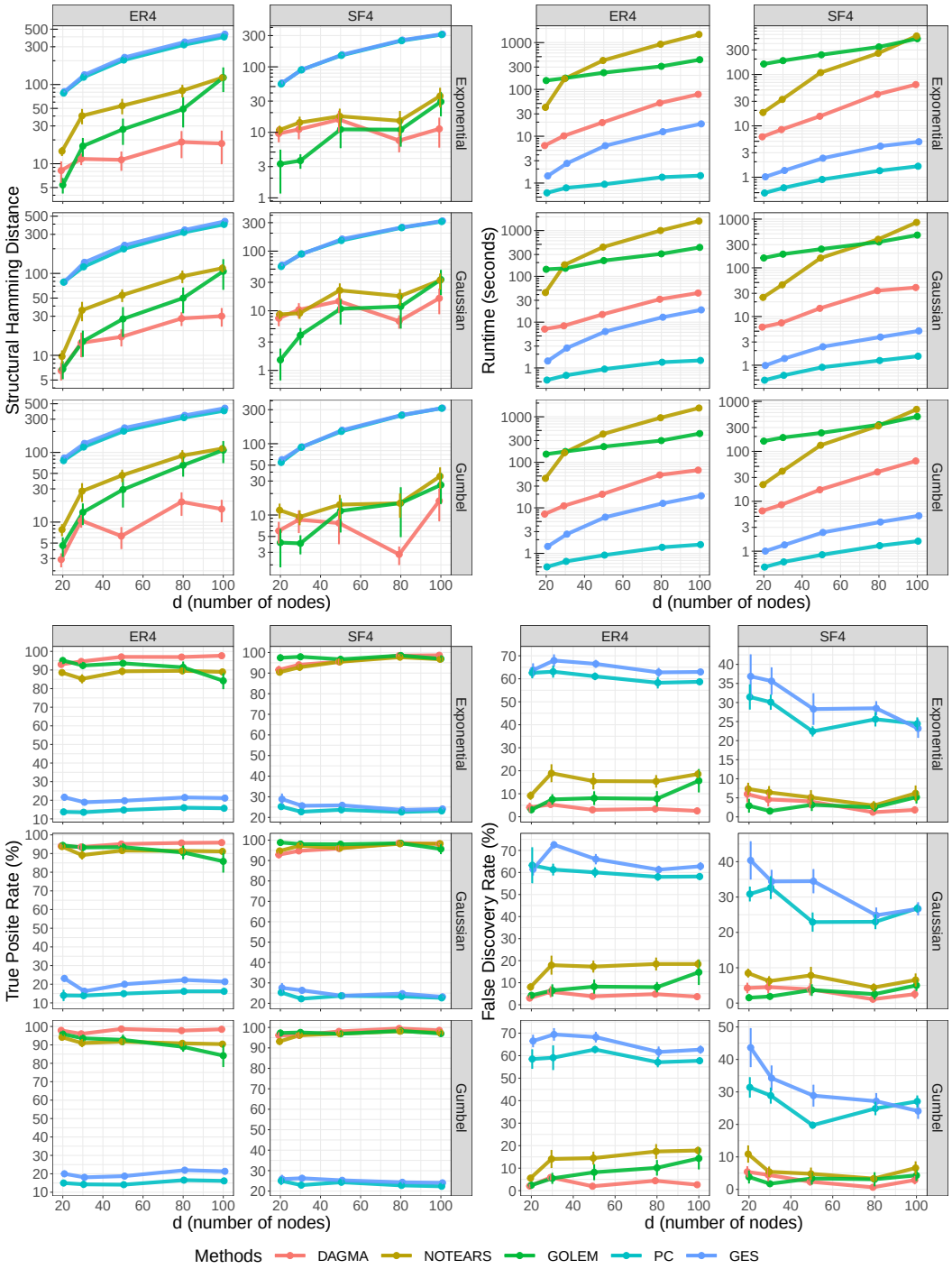


Figure 8: SHD, Runtime, TPR and FDR of all methods for different graph types (ER4, SF4) and different noise types (Gaussian, Gumbel, exponential). In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.1.1, and a summary is provided in Table 1.

C.1.3 DAGMA vs GOLEM in Sparser Graphs

We note that in the work by Ng et al. [37], the authors performed experiments on large number of nodes **only for ER2 graphs**, that is, sparser graphs. It was reported in Ng et al. [37] that their

Table 2: Summary of performances (SHD and runtime) of the most competitive methods. Each metric was averaged across different graph and noise types. Finally, the errors denote 95% confidence intervals on 10 repetitions.

Method	d	SHD	Runtime (hours)
DAGMA	200	37.77±8.80	0.06± 0.00
	300	86.65±19.63	0.13± 0.00
	500	211.68±78.88	0.37±0.02
	800	285.90±56.89	1.15±0.06
	1000	473.70±101.63	2.09±0.08
GOLEM	200	215.68±66.91	0.32±0.00
	300	552.72±113.60	0.73±0.02
	500	1390.43±213.77	2.32±0.04
	800	2919.14±191.38	7.13±0.10
	1000	4083.03±157.58	12.50±0.22
NOTEARS	200	105.58±24.19	1.28±0.14
	300	217.73±48.30	3.42±0.28
	500	441.72±93.01	10.84±0.62
	800	829.08±118.10	28.79±1.22
	1000	–	> 36

GOLEM method performed reasonably well for large number of nodes. However, as shown in Figure 9, for denser graphs such as ER4 and SF4, the performance of GOLEM degrades very fast as d increases. In this section, we experiment with the same graph model as in [37], i.e., ER2, for $d \in \{200, 300, 500, 800, 1000, 2000\}$ We note that even though GOLEM is competitive in this regime, DAGMA still obtains *significant* improvements.

DAGMA was run under the same setting described in Section C.1.1. The results are reported in Figure 10. Here we note in particular that for $d = 1000$, DAGMA obtains an **improvement** of 22.5% in **SHD**, and runs 8.5 times **faster** than GOLEM, even though the latter is customized for linear models. For $d = 2000$, GOLEM took more than 36 hours per simulation, hence, we could not report its performance.

C.1.4 DAGMA vs NOTEARS and GOLEM in Denser Graphs

For completeness, we run experiments on a denser graph type such as ER6 with Gaussian noise, for $d \in \{20, 40, 60, 80, 100\}$. DAGMA was run under the same setting described in Section C.1.1. The results are reported in Figure 11. Here we note in particular that for $d = 100$, DAGMA obtains an **improvement** of 73.1% and 44.5% in **SHD** against GOLEM and NOTEARS, respectively; also, DAGMA runs 4.8 and 20.6 times **faster** than GOLEM and NOTEARS, respectively. Finally, we note that even in the regime of small number of variables, GOLEM's performance degrades very fast for denser graphs, while DAGMA's performance remains the best among the three methods.

C.2 SEM: Non-Linear Setting

C.2.1 Logistic Model

Given the ground-truth DAG $B \in \{0, 1\}^{d \times d}$ from an ER4 graph, we assigned edge weights independently from $\text{Unif}\left([-2, -0.5] \cup [0.5, 2]\right)$ to obtain a weight matrix $W \in \mathbb{R}^{d \times d}$. Given W, we sampled $X_j = \text{Bernoulli}(\exp(w_j^\top X)/(1 + \exp(w_j^\top X))), \forall j \in [d]$. Based on this model, we generated random datasets $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ by generating the i.i.d. rows. For each simulation, we generated $n = 5000$ samples for graphs with $d \in \{20, 40, 60, 80, 100, 200, 400, 600, 800, 1000\}$ nodes.

To measure the quality of a model, we use the log-likelihood loss

$$Q(f, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^{d} \mathbf{1}_n^\top \left(\log(\mathbf{1}_n + \exp(f_i(\mathbf{X}))) - \mathbf{x}_i \circ f_i(\mathbf{X})\right). \tag{9}$$

The implementation details of the baselines are listed below:

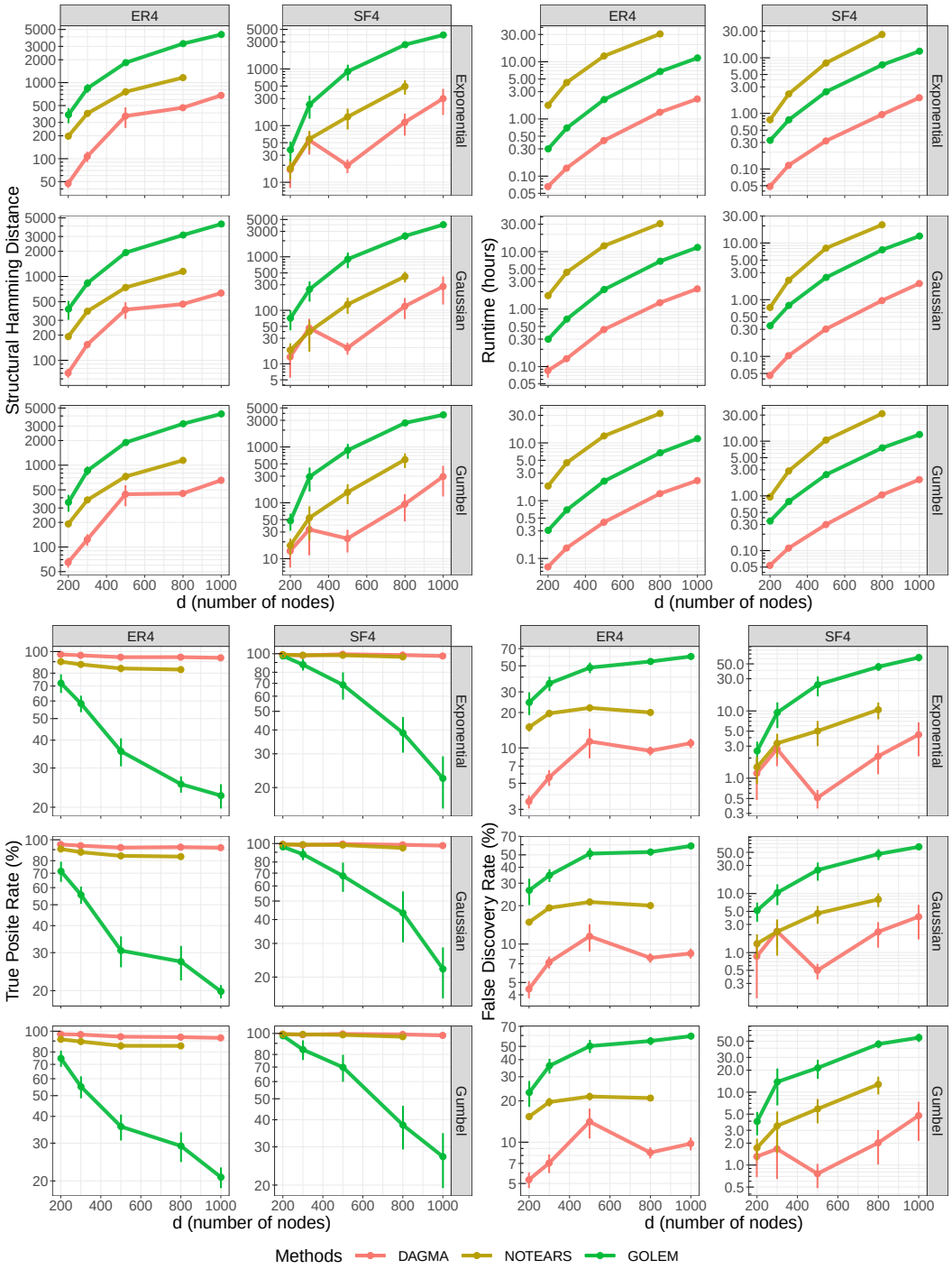


Figure 9: SHD, Runtime, TPR and FDR of competitive methods for different graph types (ER4, SF4) and different noise types (Gaussian, Gumbel, exponential). In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.1.2, and a summary is provided in Table 2.

- GES (specifically, the FGES algorithm in [47]) and PC [50] are standard baselines for structure learning. Their implementation is based on the `py-causal` package, available at `https://github.com/bd2kccd/py-causal`. The exact set of hyperparameters used are:

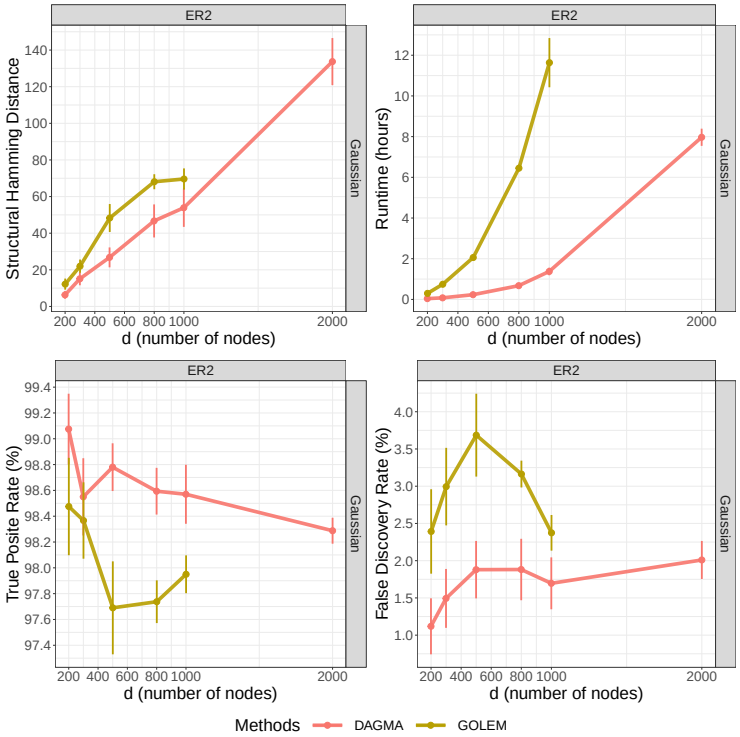


Figure 10: SHD, Runtime, TPR and FDR of DAGMA and GOLEM for a graph type ER2 with Gaussian noise. In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.1.3

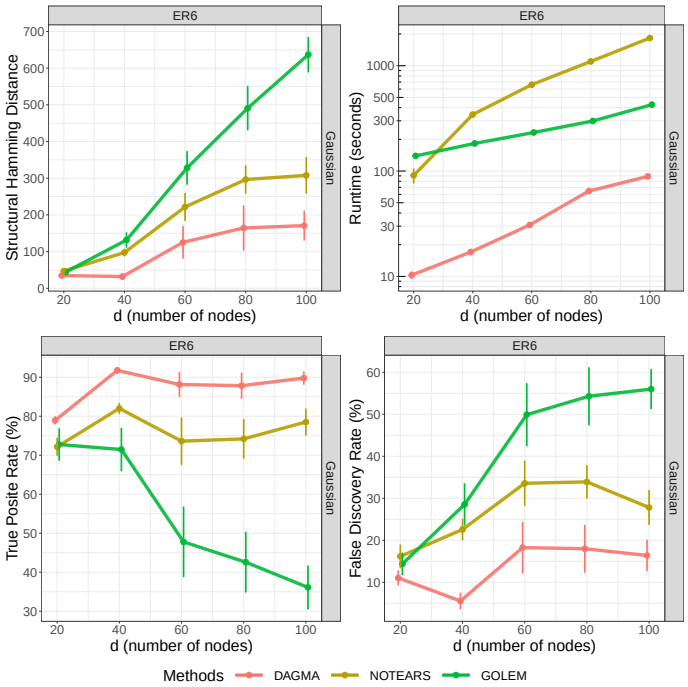


Figure 11: SHD, Runtime, TPR and FDR of DAGMA, GOLEM, and NOTEARS for a graph type ER6 with Gaussian noise. In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.1.4

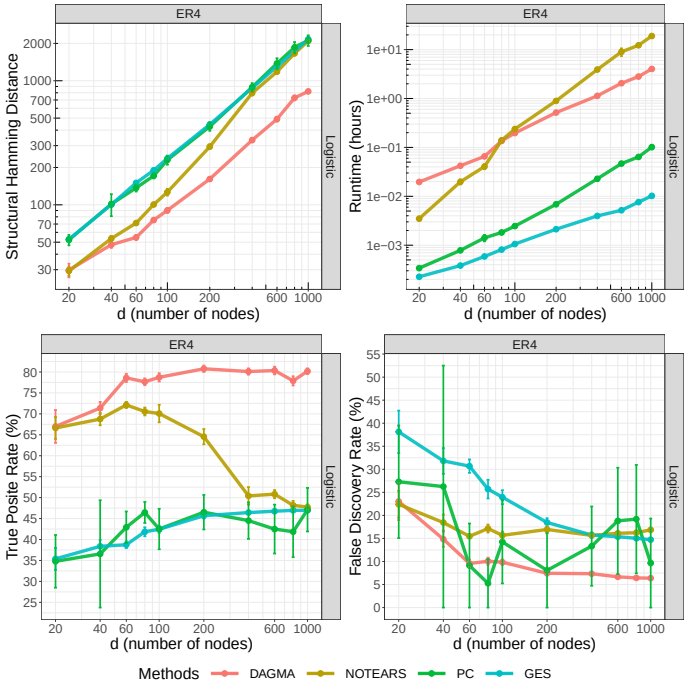


Figure 12: SHD, Runtime, TPR and FDR of all methods for a graph type ER4 and logistic model. In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.2.1

- For PC: `testId = 'disc-bic-test', depth = 4, fasRule = 2, dataType = 'discrete', conflictRule = 1, concurrentFAS = True, useMaxPOrientationHeuristic = True`.
- For GES: `scoreId = 'bdeu-score', maxDegree = 5, dataType = 'discrete', faithfulnessAssumed = False`.

- The NOTEARS method in Zheng et al. [57] was implemented using the author's Python code available at: `https://github.com/xunzheng/notears`. Its score function is also the log-likelihood loss as defined in eq.(9). For the ℓ_1 coefficient, for a fair comparison, we use the same value used for DAGMA. For the rest of hyperparameters, we use their default values.

We use the following setting for DAGMA (Algorithm 1): Number of iterations $T = 4$, initial central path coefficient $\mu^{(0)} = 10$, decay factor $\alpha = 0.1$, ℓ_1 coefficient $\beta_1 = 0.01$, log-det parameter $s = \{1, .9, .8, .7\}$. For each problem in line 3 of Algorithm 1, we implement an adaptive gradient method using the ADAM optimizer [24]. The hyperparameters for ADAM are: Learning rate of 3×10^{-4}, and $(\beta_1, \beta_2) = (0.99, 0.999)$. For $t = \{0, 1, 2\}$, we run ADAM for 10^4 iterations or until the loss converges, whichever comes first. For $t = 3$, we run ADAM for 5×10^4 iterations or until the loss converges, whichever comes first. We consider that the loss converges if the relative error between subsequent iterations is less than 10^{-6}. Finally, as in [57, 58, 37], a final thresholding step is performed as it was shown to help reduce the number of false discoveries. For all cases, we use a threshold of 0.3.

The results are shown in Figure 12. We note that for $d = 1000$, DAGMA obtains an **improvement** of 60% in **SHD** and runs 4.8 times **faster** than NOTEARS. Finally, we note that GOLEM is not considered for the nonlinear models as it only works for linear ones.

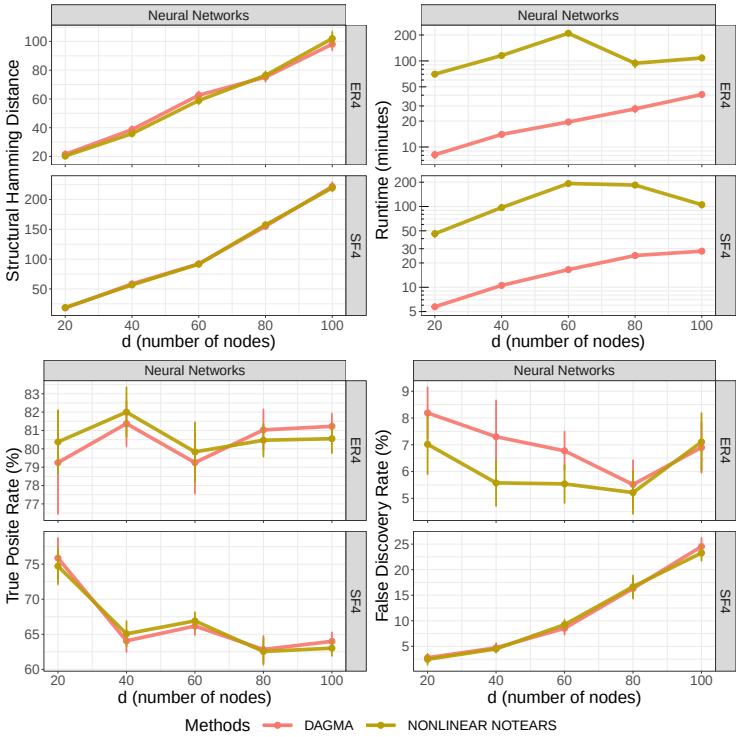


Figure 13: SHD, Runtime, TPR and FDR of all methods for a graph type ER4 and logistic model. In all cases, lower is better *except* for the TPR. Error bars represent standard errors over 10 simulations. More details are given in Section C.2.2

C.2.2 Neural Network Model

We mainly follow the nonlinear setting of Zheng et al. [58]. That is, given a ground-truth graph G, we simulate the SEM:

$$X_j = f_j(X_{\text{pa}(j)}) + Z_j, \forall j \in [d],$$

where $Z_j \sim \mathcal{N}(0, 1)$ is a standard Gaussian noise. Here f_j is a randomly initialized multilayer perceptron (MLP) with one hidden layer of size 100 and sigmoid activation. Similar to previous experiments, we generate a dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, with $n = 1000$ i.i.d. samples.

For this setting, we only compare to NONLINEAR NOTEARS [58]. We refer the reader to [58] for a comprehensive comparison with other baselines. For NONLINEAR NOTEARS and DAGMA, each f_θ is modeled by a MLP with one hidden layer of size 10 and sigmoid activation. In contrast to the original implementation of NONLINEAR NOTEARS [58] which uses the square loss, we use the log-likelihood as in [9] as we observe better performances for both methods.

We use the following setting for DAGMA (Algorithm 1): Number of iterations $T = 4$, initial central path coefficient $\mu^{(0)} = 0.1$, decay factor $\alpha = 0.1$, ℓ_1 coefficient $\beta_1 = 0.02$, log-det parameter $s = 1$. For each problem in line 3 of Algorithm 1, we implement an adaptive gradient method using the ADAM optimizer [24]. The hyperparameters for ADAM are: Learning rate of 2×10^{-4}, and $(\beta_1, \beta_2) = (0.99, 0.999)$. For $t = \{0, 1, 2\}$, we run ADAM for 7×10^4 iterations or until the loss converges, whichever comes first. For $t = 3$, we run ADAM for 8×10^4 iterations or until the loss converges, whichever comes first. We consider that the loss converges if the relative error between subsequent iterations is less than 10^{-6}. Finally, as in [57, 58, 37], a final thresholding step is performed as it was shown to help reduce the number of false discoveries. For all cases, we use a threshold of 0.3.

The results are shown in Figure 13. We note that DAGMA and NOTEARS obtain similar performances in SHD; however, DAGMA can obtain 3x to 10x speedups over NONLINEAR NOTEARS. Finally, we note that GOLEM is not considered for the nonlinear models as it only works for linear ones.

D Broader Impacts

A potential misuse of this type of work would be to purposely (or not) run the method proposed on a dataset that is biased. Since we do not formally deal with inherent biases in the dataset (e.g., unfairness due to selection bias), it is possible to learn relationships that are not present in reality. A user can then (un)intentionally report a result incorrectly claiming to have found the cause of a certain variable, thus, creating misinformation.