NOTEARS
A Stuctural Learning Algorithm

Marco Sousa-Poza

April 13, 2024

1 Introduction

This report focuses on the implementation and assessment of the algorithm called *"**N**on-combinatorial **O**ptimisation via **T**race **E**xponential and **A**ugmented lag**r**angian for **S**tructure learning"* (*NOTEARS*), a technique developed by Zheng et. al. to solve the NP-hard problem of learning Bayesian networks from data.[11] Bayesian networks, which are graphical models that represent a set of variables and their conditional dependencies via a directed acyclic graph (DAG), are widely used across various domains such as biology[9], genetics[10], machine learning[6], and causal inference[7] underscoring the importance of devising efficient and dependable methods for their discovery. Conventionally, identifying Bayesian networks from data has involved a combinatorial optimisation challenge that searched for causal influences between variables of a given dataset. Formally, this can be defined as finding a causal network, also known as a directed acyclic graph (DAG), that best explains the data:

$$\begin{aligned} &\min_{G \in \mathcal{G}} D(G, X) \\ &\text{subject to } G \in \textbf{DAGs} \end{aligned} \tag{1}$$

Here, $\mathcal{G}$ represents the set of directed graphs, $D : \mathcal{G} \times \mathbb{R}^{d,n} \to \mathbb{R}$ denotes a discrete score function computed over data with n observations with d dimensions $X \in \mathbb{R}^{d \times n}$, and $\textbf{DAGs} \subset \mathcal{G}$ specifies the subset of acyclic graphs. The difficulty in solving Equation 1 is to find an effective global search strategy for the seach space of feasible solutions $G \in \textbf{DAGs}$ that grows super exponentially in the number of variables in the dataset.

NOTEARS proposes a novel approach by converting the combinatorial optimisation into a continuous optimisation problem. This is done by recasting both $\min_{G \in \mathcal{G}} D(G, X)$ and the condition $G \in \textbf{DAGs}$ into smooth functions, allowing for efficient resolution through gradient-based optimisation techniques that no longer require a custom search strategy for navigating through the combinatorial search space to find an optimal solution for G.

The essence of this report is to thoroughly examine the *NOTEARS* algorithm, beginning with a concise overview of structural learning and the significance of graphs in Bayesian

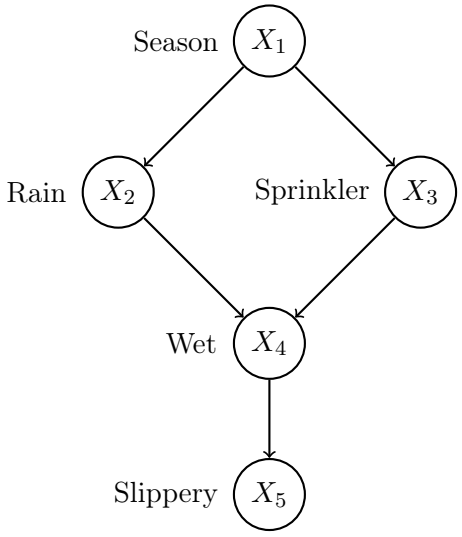
learning. It further discusses how *NOTEARS* adeptly tackles the structural learning challenge by transforming the discrete optimisation problem into a continuous, smooth optimisation challenge.

Subsequently, the document presents a sequence of experiments aimed at evaluating *NOTEARS*' performance, comparing it against the findings in the original study, and identifying potential enhancements through model averaging and the fine-tuning of hyperparameters. These experiments are intended to ascertain the practical efficacy of *NOTEARS* and explore its capacity to improve the precision and efficiency of discovering Bayesian networks.

2 Structural Learning

2.1 Directed Graphs

A directed graph $G = (V, E)$ is a discrete structure used to represent relationships between elements of a finite, discrete set known as vertices $V = \{v_1, v_2, \ldots, v_n\}$ through directed edges $E \subseteq V \times V$. Figure 1 illustrates a graph's visualisation, with vertices represented by nodes and directed edges by arrows.



$$G = (V, E)$$
$$V = \{X_1, X_2, X_3, X_4, X_5\}$$
$$E = \left\{ \begin{array}{c} (X_1, X_2), (X_1, X_3), (X_2, X_4), \\ (X_3, X_4), (X_4, X_5) \end{array} \right\}$$

Figure 1: Example Graph

Graphs are characterised by various terms. To aid communication, a brief overview of these terms is presented. Each informal definition is paralleled by a mathematical definition provided in Appendix A.

An **edge** is an arrow connecting two vertices e.g. $A \to B$ would be one edge. We call A and B **adjacent** for having an edge between them. Furthermore, A is the **parent** of B and

conversely B is the **child** of A. Consistent with the analogy of heritage there are also the terms **ancestors** and **descendants** defined by recursively gathering parents and children respectively.

A **path** in a directed graph is a sequence of adjacent vertices connected by directed edges, with traversal following the edge direction from one vertex to the next. If there is a way to travel from vertex u to vertex v by traversing these directed edges, a path exists between them. An **undirected path** is defined by allowing movement along edges without regard to their direction, enabling travel between two vertices irrespective of the edge's orientation. **Acyclic graphs** are those from which no path leads from any node back to itself.

For mathematical convenience, a graph can also be defined via an **adjacency matrix**. This matrix W has dimensions $\{0,1\}^{d\times d}$ for a given graph $G=(V,E)$, is defined as:

$$W_{i,j} = \begin{cases} 1 & \text{if } (i,j)\in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The adjacency matrix serves as a potent tool for graph analysis, with matrix operations corresponding to meaningful graph operations. For instance, the n-th power of an adjacency matrix W^n encapsulates all length-n paths within a graph. Each element $W^n_{i,j}$ counts the number of n-step paths from vertex i to j, calculated by summing over intermediary vertices that connect i to j through exactly n intermediate vertices. This is illustrated in Figure 2. This property provides an alternative definition for acyclicity, proving beneficial later in the report:

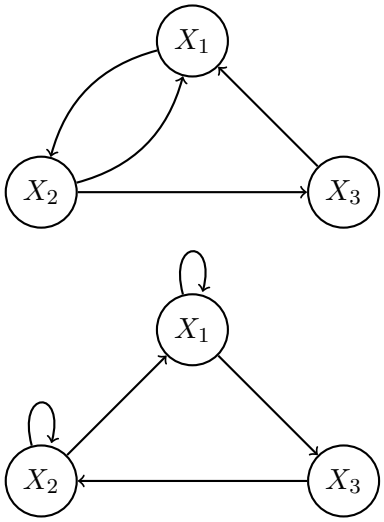
$$g(W)\in \textbf{DAGs} \iff \sum_{i=1}^{d} tr(W^i) = 0 \tag{3}$$

Where $g:\{0,1\}^{d,d}\to\mathcal{G}$ transforms the adjacency matrix to its corresponding graph, $tr(W)$ is the trace of matrix W and $\textbf{DAGS}\subset\mathcal{G}$ is the set of all acyclic graphs. In essence, equation 3 illustrates that for a directed graph to qualify as a DAG, it must not contain a path of any length that starts and ends at the same node.

2.2 Probability and DAGs

Bayesian networks represent a powerful framework for probabilistic reasoning and understanding the dependencies among a set of variables. By modeling the variables as vertices in a directed acyclic graph (DAG) and the dependencies between these variables as edges, Bayesian networks can capture complex relationships within data. This structure not only helps in visualizing the conditional dependencies but also simplifies the computation of joint probabilities.

The vertices in a Bayesian network graph correspond to random variables, denoted as $V=\{X_1,X_2,\ldots,X_d\}$ for a graph $G=(V,E)$. A key feature of using a DAG for representing these variables is its ability to encode conditional independencies, which directly influences how we compute the joint probability distribution $P(V)=P(X_1,X_2,\ldots,X_d)$. The Markov



$$\iff \quad W^1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\iff \quad W^2 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 2: Size 2 Path Calculation

condition, encoded in these networks, states that a variable is conditionally independent of its non-descendants given its parents. This allows the joint probability distribution over the entire set of variables to be decomposed into a product of simpler conditional probability distributions:

$$P(X_1, X_2, \ldots, X_d) = \prod_i P(X_i \,|\, \mathrm{Pa}(X_i)) \tag{4}$$

where $\mathrm{Pa}(X_i)$ denotes the set of parent variables of X_i in the graph.

This decomposition is fundamental because it means that instead of calculating a potentially intractable full joint probability distribution, we can work with smaller, more manageable conditional distributions. This significantly reduces the complexity of probabilistic calculations and is especially useful in reasoning under uncertainty.

For instance, considering the question "What is the probability that it rained given that it is slippery and the sprinkler was off?" in the context of a Bayesian network like the one described earlier, we leverage these conditional independencies. The strength of Bayesian networks lies in their ability to reason under uncertainty given prior information. Given the structure of the network and the known states (e.g., Slippery is true and Sprinkler is off), we can efficiently compute the desired probability by focusing only on the relevant parts of the network and applying Bayes' theorem, along with the conditional probabilities provided by the network. This process involves considering the relationships encoded in the graph—how "Slippery" depends on "Wet," which in turn depends on both "Rain" and "Sprinkler," and these dependencies allow us to infer the probability of "Rain" given the observed conditions.

Bayesian networks' ability to encode conditional independencies and to decompose complex joint distributions into products of simpler conditional distributions is what empowers

them to efficiently reason under uncertainty, given prior information. This makes them invaluable in various application areas, including medical diagnosis [5], genetics research [1], and numerous other fields where making decisions under uncertainty is crucial.

2.3 Learning from Data

Oftentimes, Bayesian networks are (partially) given by domain experts [1, 4, 5]. In cases where such domain expertise is not available, we resort to inferring the graph structure from data. This process, aimed at identifying conditional independencies within a dataset, is known as *structural learning*. Structural learning can be formally described as a constrained optimisation problem.

Definition. *Let* $\mathbf{X}$ *represent our data matrix in* $\mathbb{R}^{d\times n}$*, and let* $W \in \mathbb{R}^{d\times d}$ *denote a directed graph represented as an adjacency matrix. Furthermore, define a loss function* $F(W, \mathbf{X}) : \mathbb{R}^{d\times d} \times \mathbb{R}^{n\times d} \to \mathbb{R}$ *that assesses how well the graph* W *explains our data* $\mathbf{X}$ *(further specified in Section 3.2). Then, structural learning can be defined as the optimization problem:*

$$\begin{aligned} &\min_{W\in\{0,1\}^{d\times d}} F(W, \mathbf{X}) \\ &\textit{subject to } G(W) \in \mathbf{DAGs} \end{aligned} \tag{5}$$

where $\mathbf{DAGs} \subset \mathcal{G}$ *represents the set of all Directed Acyclic Graphs, and* $g : \mathbb{R}^{d\times d} \to \}$ *is a function that transforms the adjacency matrix* W *into a graph.*

This definition underscores the goal of finding a directed graph that best explains the data under the constraint that the graph must be acyclic.

Addressing this as an optimisation problem presents several notable challenges:

(1) The definition and specification of the loss function F are crucial, as there is no universal method to functionally formalise causality. For example, a causal graph like $X_1 \to X_2 \leftarrow X_3$ could represent relationships such as $X_1 = X_2 \cdot X_3$ and $X_1 = X_2 + X_3 + b$ for some constant $b \in \mathbb{R}^d$. Thus, the system's relational structure is assumed.

(2) Verifying that an adjacency matrix W represents an acyclic graph, as required by Equation 5, is not a numerical task but rather an algorithmic check. Although this check can be performed efficiently in $\mathcal{O}(d^2)$ time, the function for this check is discrete and not differentiable, complicating the optimisation process toward satisfying the DAG constraint.

(3) The computational challenge is compounded by the fact that the number of possible DAGs increases super-exponentially with the number of nodes. The recurrence relation for the number of DAGs of size d, as documented in the OEIS [8], is denoted by $a(d)$ for $d \in \mathbb{Z}^+$, where

$$a(d) = \begin{cases} 1 & \text{if } d = 0 \\ \sum_{k=1}^{d} (-1)^{k+1} \cdot \binom{d}{k} \cdot 2^{k(d-k)} \cdot a(d-k) & \text{otherwise} \end{cases} \tag{6}$$

A conservative lower bound is given by the number of all upper triangular adjacency matrices which is a subset of the **DAGs** set. The number of d-sized binary upper triangular matrices also grows super-exponentially at a rate of $2^{\frac{d(d-1)}{2}}$ giving a good indication of how quickly $a(d)$ grows.

3 NOTEARS

3.1 General Idea

The *NOTEARS* algorithm introduces a novel approach for learning the structure of DAGs directly from data, overcoming the limitations of traditional structure learning methods outlined in Section 2.3. This algorithm reformulates the optimization problem presented in Equation 5 into a continuous optimization task. It employs a smooth, differentiable constraint to ensure the acyclicity of the learned graph, thereby enabling the use of gradient-based optimization techniques. This acyclicity constraint is ingeniously integrated within the optimization objective, facilitating the efficient enforcement of the DAG structure without necessitating discrete checks. As a result, *NOTEARS* significantly lowers the computational complexity tied to structure learning in DAGs, enhancing scalability to larger datasets.

3.2 Loss Function

Defining a continuous loss function F is a crucial step in transitioning to a continuous constraint optimization problem. Initially, in the formulation of the problem presented in Equation 5, the loss function appears discrete due to the discrete nature of the input graph $W \in \{0,1\}^{d\times d}$. To circumvent this, the authors expand the domain of the adjacency matrix from $\{0,1\}^{d\times d}$ to $\mathbb{R}^{d\times d}$, thus allowing edges to assume any real value. An edge is considered present if $W_{i,j} \neq 0$. In this context, in a slight deviation from conventional notation, edges not only signify the existence of a causal connection but also its magnitude.

With the revised definition of the adjacency matrix, it becomes possible to define any differentiable function with the domain $F : \mathbb{R}^{d\times d} \times \mathbb{R}^{n\times d} \to \mathbb{R}$ as the loss function. For example, assuming linear dependencies among variables ($X_i = \sum_{X_j \in \mathrm{Pa}(X_j)} X_j W_{i,j} + \epsilon_i$), a linear system of equation loss can be utilized:

$$F(W, X) = \frac{1}{2n}||X - XW||_F^2 \tag{7}$$

where $||\cdot||_F$ denotes the Frobenius norm. However, it is also feasible to employ loss functions suited to different domains of X, such as a logistic loss function.

To promote the generation of sparse graphs, the loss function can be augmented with l_1 regularization:

$$F(W, X) = \frac{1}{2n}||X - XW||_F^2 + \lambda||W||_1 \tag{8}$$

where λ is a regularisation parameter controlling the strength of regularisation, and $|\cdot|_1$ is the l_1 norm, defined as $||W||_1 = \sum_{i,j} |W_{i,j}|$.

3.3 Acyclicity Constraint

The final step in formulating a continuous optimization problem involves defining a continuous function $h : \mathbb{R}^{d\times d} \to \mathbb{R}$ that evaluates the presence of cycles in W. The authors of the *NOTEARS* paper specify four key requirements for the function h:

(a) $h(W) = 0$ if and only if W is acyclic;

(b) The function h should quantify the "DAG-ness" of a graph, meaning a graph W_1 with shorter cycles should be considered "worse" than a graph W_2 with longer cycles, i.e., $h(W_1) > h(W_2)$;

(c) h must be smooth;

(d) h must be differentiable with respect to W and computationally efficient.

This function can be derived from the DAG constraint definition presented earlier. Extending the domain to continuous adjacency matrices gives us $h : \mathbb{R}^{d\times d} \to \mathbb{R}$ defined as:

$$h(W) = \sum_{i=1}^{d} \operatorname{tr}\left((W \circ W)^i\right) = \sum_{i=0}^{d} \operatorname{tr}\left((W \circ W)^i\right) - d \tag{9}$$

where $\circ$ denotes the Hadamard product, effectively transforming all weighted edges in W into positive edges.

This formulation fulfills the acyclicity, smoothness and differentiability requirements. However, it does not satisfy the criterion for "DAG-ness" since cycles of different lengths influence h equally. To address this, a scaling factor is introduced:

$$h(W) = \sum_{i=1}^{d} \operatorname{tr}\left(\frac{1}{i!}(W \circ W)^i\right) - d = e^{W\circ W} - d \tag{10}$$

This adjusted function fulfills all specified requirements. The derivative of $h(W)$ is:

$$\Delta_W h(W) = \left(e^{W\circ W}\right)^T 2W \tag{11}$$

3.4 Dual Formalisation

Putting the pieces together from Section 3.2 and 3.3 the final altered optimisation problem is defined as:

$$\begin{aligned} &\min_{W\in\mathbb{R}^{d,d}} F(W,X) + \lambda|W|_1 \\ &\text{subject to } h(W) = 0 \end{aligned} \tag{12}$$

For solving this optimisation problem the authors of the paper propose to first alter Equation 12 using a quadratic penalty term

$$\begin{aligned} &\min_{W\in\mathbb{R}^{d,d}} F(W,X) + \lambda|W|_1 + \tfrac{\rho}{2}\left(h(W)\right)^2 \\ &\text{subject to } h(W) = 0 \end{aligned} \tag{13}$$

with $\rho > 0$. And finally putting it into its augmented Lagrangian dual formalisation:

$$\begin{aligned} D(\alpha) &= \min_{W \in \mathbb{R}^{d,d}} L^{\rho}(W, \alpha) \\ \text{where} \quad L^{\rho}(W, \alpha) &= F(W, X) + \lambda |W|_1 + \frac{\rho}{2} \left(h(W)\right)^2 + \alpha h(W) \end{aligned} \tag{14}$$

The goal is to find a local minimiser to the problem:

$$\max_{\alpha \in \mathbb{R}} D(\alpha) \tag{15}$$

This problem can be solved using dual ascent, which is the process of iteratively optimising with respect to the dual and primal variable. Formally we can write

$$W_{\alpha^{(t)}}^{(t+1)} = \min_{W \in \mathbb{R}^{d,d}} L^{\rho}(W, \alpha^{(t)}) \tag{16}$$

$$\alpha^{(t+1)} = \alpha^{(t)} + \mu \cdot \frac{\mathrm{d}}{\mathrm{d}\alpha} \left[L^{\rho}(W_{\alpha^{(t)}}^{(t+1)}, \alpha) \right]_{\alpha = \alpha^{(t)}} \tag{17}$$

where μ is the learning rate. The dual variable is therefore optimised using gradient ascent. Since the augmented dual function depends linearly on the dual variable α the paper proposes the update rule:

$$\alpha^{(t+1)} = \alpha^{(t)} + \rho h(W_{\alpha^{(t)}}^{(t+1)}) \tag{18}$$

The paper gives no clear reasoning behind choosing the learning rate being equal to the quadratic penalty term i.e. $\mu = \rho$, although experiments show that given ρ large enough the dual problem will converge to a local minimum satisfying the DAG constraint. To find the local minimser $W_{\alpha^{(t)}}^{(t+1)}$ any descent method method can be used, although the paper proposes to use a quasi Newtonian optimisation technique for faster convergence. Note that due to using $l1$ regularisation not being differentiable at 0 slight care needs to be taken. The authors refer to a paper by Zhong et. al. that is used to solve optimisation problems using l_1 regularisation.[12] The specifics are outside the scope of this report.

3.5 Final Algorithm

Given the specification of the dual function and the methodology for solving it, only a few implementation details need to be highlighted:

1. **Progress Rate:** To ensure faster convergence towards a solution satisfying the DAG constraint, we require that before optimizing α, as specified in Equation 18, $h(W_{\alpha^{(t)}}^{(t+1)})$ decreases by a certain fraction compared to $h(W_{\alpha^{(t-1)}}^{(t)})$. This is achieved by incrementally increasing the quadratic penalty term ρ until

   $$h(W_{\alpha^{(t)}}^{(t+1)}) < c \cdot h(W_{\alpha^{(t-1)}}^{(t)}) \quad \text{with} \quad c \in (0, 1) \tag{19}$$

   The rationale behind increasing ρ to meet this condition is explained in the following point.

2. **Quadratic Penalty Term:** The role of ρ as both a learning rate and a quadratic penalty term is crucial yet not extensively discussed in the original paper. ρ ensures that the optimization process adequately emphasizes the DAG constraint; a larger ρ increases the gradient of $\Delta_W \rho[h(W)]^2$ in the augmented dual function, facilitating a quicker descent towards a DAG-compliant solution. Consequently, inflating ρ helps meet the progress condition mentioned previously. The rate at which ρ should be inflated is not specified; however, in the implementation used by the authors, ρ is exponentially increased according to the update rule $\rho \leftarrow \rho^{10}$.

3. **Termination Condition:** Theoretically, the dual ascent method concludes when $h(W_\alpha^*) = 0$, indicating a local optimum for both dual and primal variables has been reached. Due to limitations in machine precision, it is advisable to terminate the process when $h(W_\alpha^*) < \epsilon$ for a sufficiently small $\epsilon > 0$. The authors note that this typically occurs within approximately 10 steps of dual ascent, i.e., $t = 10$.

4. **Final Result:** As per the dual formulation, the *NOTEARS* algorithm identifies not only the local minimiser of the dual variable α^* but also a local minimiser $W^* = \min_{W \in \mathbb{R}^{d \times d}} L^\rho(W, \alpha^*)$. This solution remains within the real number domain. To derive a graph representation, i.e., $G^* = G(W^*)$, we define $G : \mathbb{R}^{d \times d} \to V \times E$ as follows:

$$\begin{aligned} G(W) &= (V, E) \\ V &= \{X_i \,|\, i = 1, 2, \ldots, d\} \\ E &= \{(X_i, X_j) \,|\, i, j = 1, 2, \ldots, d \text{ if } |W_{i,j}| > \omega\} \end{aligned} \tag{20}$$

   where $\omega > 0$ is a threshold value that needs careful selection. The choice of ω is critical; setting it too high may exclude significant edges, whereas too low a value might include excessive edges, potentially breaching the DAG constraint if ϵ is not adequately small. Figure 3 offers a visualisation one can use to determine a good value for ω.

The complete algorithm is formalised as pseudo code in Algorithm 1.

4 Implementation

4.1 NOTEARS Implementation

To evaluate and understand the *NOTEARS* algorithm, I replicated its implementation in Python. Although fundamentally similar, some simplifications were adopted. Unlike the original *NOTEARS*, which employs an extended adjacency matrix covering all real numbers as discussed in Section 3.2, this implementation is restricted to non-negative real numbers. This restriction simplifies the dual problem solution from Equation 14, utilising the following sub-gradient for the l_1 regularisation term's non-smooth aspect:

$$\frac{\mathrm{d}}{\mathrm{d}W_{i,j}}|W|1 = \lim_{h \to 0^+} \frac{|W_{i,j} + h| - |W_{i,j}|}{h} = 1 \tag{21}$$

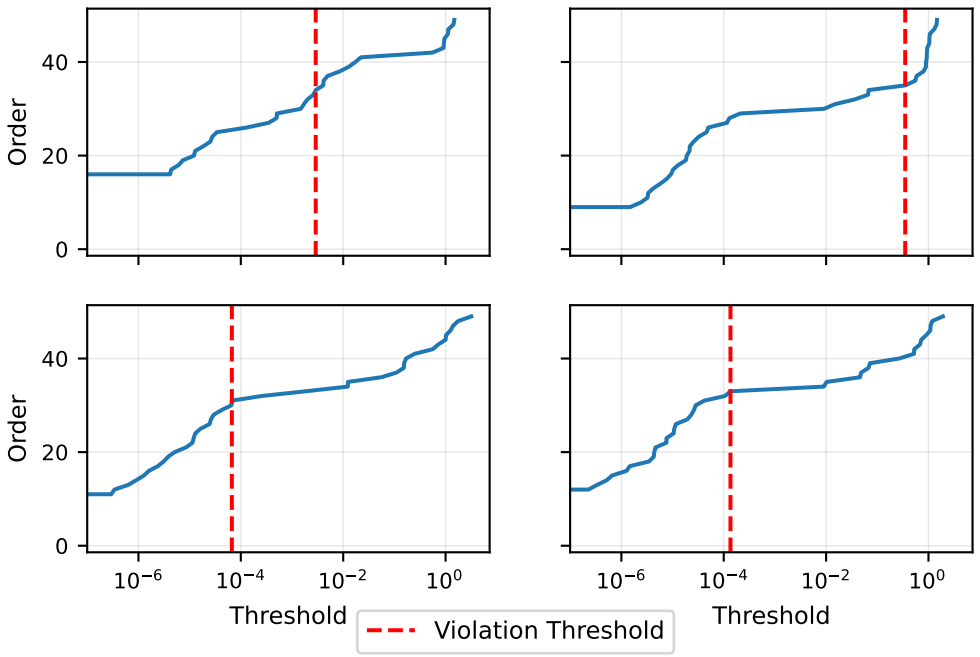


Figure 3: Depiction of what kind of setting for ω results in which order (number of edges) of the discovered graph. This figure is based on the *NOTEARS*' findings on four different randomly generated datasets. Note the logarithmic scale on the x-axis demonstrating the sensitivity of this parameter. The red bar represents the highest possible ω value after which the binary graph $G(W^*)$ is no longer a DAG.

An additional modification involves incorporating a hyper-parameter c, which determines the algorithm's progress rate. The original paper does not detail its choice, employing $c = 0.25$. This adjustment enabled performance testing at varying progress rates.

Aside from these modifications, my implementation aligns with the algorithm described in 1. The code is available on GitHub at: marcosousapoza/notears.

4.2 Data Generation

To validate my implementation, I created a function for autonomous data generation, comprising two steps:

(1) **Random DAG Generation:** Initially, a random DAG is created, serving as the data's foundation. Following the *NOTEARS* specification, a random adjacency matrix $\widetilde{W}$ is generated with edges within the realm of non-negative real numbers, determined by a probability α. Higher α values produce denser graphs, whereas lower values yield sparser ones.

Algorithm 1 NOTEARS Algorithm

Initialise α^0, $W^{(0)}_{\alpha^{(0)}}$, ρ ▷ Authors choose $\rho = 1$
repeat
  repeat
    $W^{(t+1)}_{\alpha^{(t)}} \leftarrow \min_{W \in \mathbb{R}^{d,d}} L^{\rho}(W, \alpha^{(t)})$
    if $h(W^{(t+1)}_{\alpha^{(t)}}) > c \cdot h(W^{(t)}_{\alpha^{(t)}})$ **then** ▷ Authors choose $c = 0.25$
      $\rho \leftarrow \rho^n$ ▷ Authors choose $n = 10$
    end if
  until $h(W^{(t+1)}_{\alpha^{(t)}}) > c \cdot h(W^{(t)}_{\alpha^{(t)}})$
  $\alpha^{(t+1)} \leftarrow \alpha^{(t)} + \rho h(W^{(t+1)}_{\alpha^{(t)}})$
until $h(W^{(t+1)}_{\alpha^{(t)}}) < \epsilon$

(2) **Data Generation:** Data is then produced using the adjacency matrix. Variables lacking parent nodes in the DAG W are sampled from a normal distribution ($X_i \sim \mathcal{N}(0, \sigma_i)$), and for subsequent variables, $X_j = XW + \epsilon_j$ is computed, where $\epsilon_j \sim \mathcal{N}(0, \sigma_j)$. Although the *NOTEARS* paper states the distribution of ϵ_j is arbitrary, this implementation opts for normally distributed errors for simplicity, aligning with the linear system model and corresponding loss function described in Equation 7.

5 Experiment

5.1 Metrics

To assess the performance of the *NOTEARS* algorithm different metrics are used to compare the found graph to the true graph from which the data is generated as described in Section 4.2. For notational purposes we will use W^* for the adjacency matrix *NOTEARS* inferred given the respective threshold $\omega > 0$ and $\widetilde{W}$ for the true binary adjacency matrix. All metrics are based on the following five statistics which use metrics described in table 1:
From these statistics we can form composite statistics:

- **Strucural Hamming Distance**: SHD $= FP + FN + R$. This measures how many deletions additions and reversals need to be performed in order to convert W^* into $\widetilde{W}$;
- **True Positive Rate**: TPR $= TP \ / \ (TP + FN)$;
- **False Positive Rate**: FPR $= (R + FP) \ / \ (FP + TN)$. Not to confuse with the FPR known from binary class predictions. Here R is added to the nominator to not punish reversed edges as they still give valuable insight into the relation between variables;
- **False Discovery Rate**: FDR $= (R + FP) \ / \ (FP + TP)$;

Metric	Definition
TP (True Positive)	$\sum_{i=1}^{d}\sum_{j=1}^{d}\mathcal{I}\left[W^*_{i,j} < \omega \wedge \widetilde{W}_{i,j} = 1\right]$
FP (False Positive)	$\sum_{i=1}^{d}\sum_{j=1}^{d}\mathcal{I}\left[W^*_{i,j} < \omega \wedge \widetilde{W}_{i,j} = 0\right]$
FN (False Negative)	$\sum_{i=1}^{d}\sum_{j=1}^{d}\mathcal{I}\left[W^*_{i,j} \geq \omega \wedge \widetilde{W}_{i,j} = 1\right]$
TN (True Negative)	$\sum_{i=1}^{d}\sum_{j=1}^{d}\mathcal{I}\left[W^*_{i,j} \geq \omega \wedge \widetilde{W}_{i,j} = 0\right]$
R (Reversed Edges)	$\sum_{i=1}^{d}\sum_{j=1}^{d}\mathcal{I}\left[W^*_{i,j} < \omega \wedge \widetilde{W}_{j,i} = 1\right]$

Table 1: Definition of TP, FP, FN, TN and R in the context of graph learning. The function $\mathcal{I}$ is the indicator function, W^* is the found adjacency matrix and $\widetilde{W}$ is the adjacency matrix for the true graph respctively.

5.2 Preliminary Testing

This test performs two experiments to evaluate the performance of the NOTEARS algorithm in discovering the structure of DAGs from data generated according to a linear structural equation model as explained in the previous section. To make the task challenging a random DAG with $d = 7$ nodes and $\alpha = 0.5$ is generated, resulting in a larger dense graph. For the *NOTEARS* algorithm the standard hyper parameters used in the paper with slight regularisation were chosen: $\lambda = 0.01, \epsilon = 10^{-8}, c = 0.25$. The results are depicted in Figure 4.

It is visible there that the algorithm finds most of the edges. Some mistakes are made, implying either that the model got stuck in a local minimum, or the optimal solution is not equal to the true graph. An interesting observation is also that the algorithm finds edges that are pointing in the wrong direction. This is a common problem in structural learning as it can be difficult to distinguish between cause and effect purely based on observational data. The presence of reverse edges suggests that while the algorithm is capable of identifying relationships between variables, it sometimes struggles to correctly infer the directionality of these relationships. This issue is inherent to causal discovery from observational data, where, without interventions or additional assumptions, certain causal structures are indistinguishable. These undetermined cases are known as Markov equivalence classes, where multiple DAGs can represent the same set of conditional independence relationships observable in the data.[2]

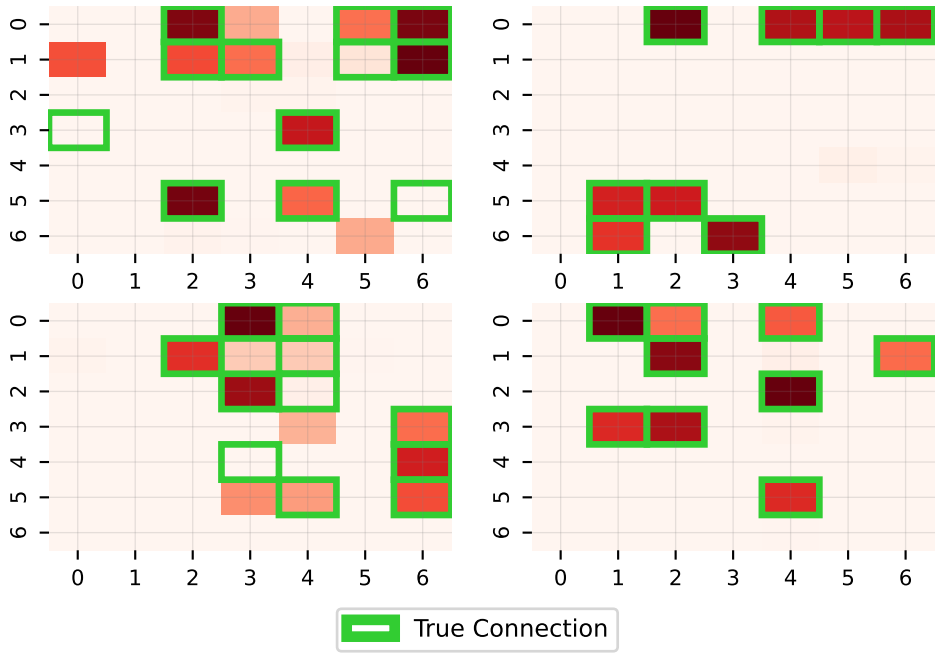


Figure 4: Preliminary test of the *NOTEARS* algorithm on four different random datasets. The coloured cells are a depiction of the magnitude of $W_{i,j}$ found by the algorithm. The true underlying graph is marked with green squares.

5.3 Progress Rate & Model Averaging

As already pointed at in Section 4.1 the progress rate specified in the original *NOTEARS* paper is not discussed. This test shows the sensitivity to this hyperparamter. Just as in the previous test, random data is generated with $d = 7$ and $\alpha = 0.5$. Similarly the hyperparameters for the *NOTEARS* algorithm are set to $\lambda = 0.01, \epsilon = 10^{-8}$ with varying progress rate $c \in (0, 1)$.

Figure 5 shows that different the progress rates can effect the performance of the algorithm as the variation in scores indicates. It demonstrates the importance of running *NOTEARS* using different progress rates; and different hyperparameters in general; to potentially find more causal connections and get more robust findings.

One method that has proven to be quite robust is to run *NOTEARS* multiple times for different hyper-parameters on the data and average over all results. That is for the graphs $W_1^*, W_2^*, \ldots, W_n^*$ found by n independent runs of *NOTEARS* we can average their results to get our final graph:

$$W^* = \frac{1}{n} \sum_{i=1}^{n} W_i^* \tag{22}$$

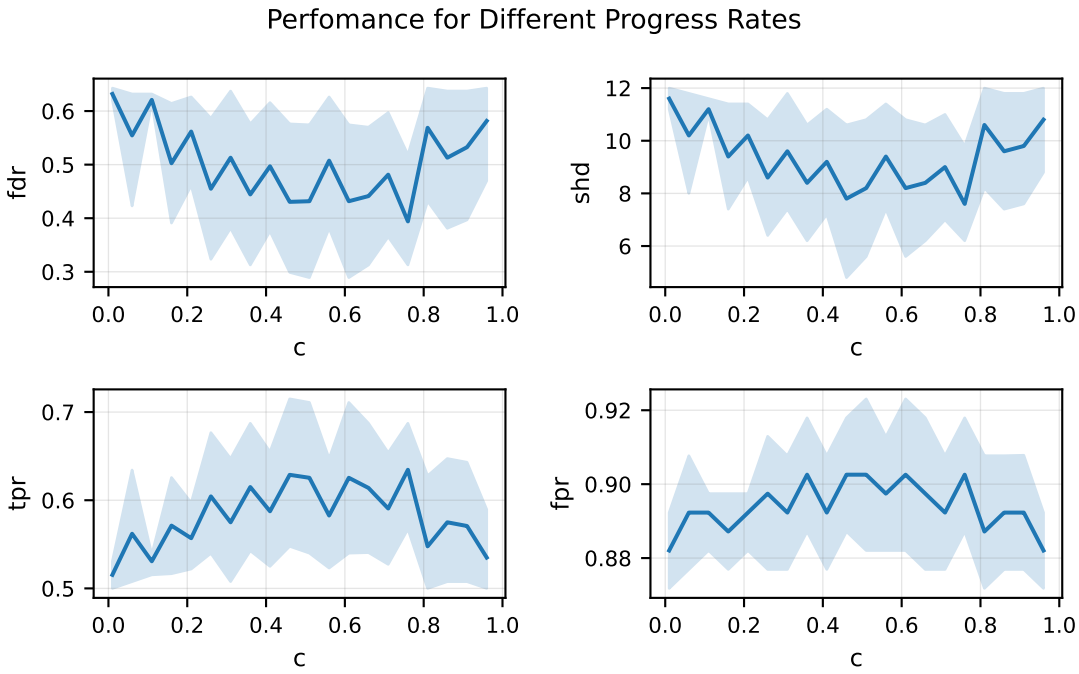


Figure 5: Experiment depicting average effect of different progress rates $c \in (0, 1)$ have on metrics TPR, FPR, FDR and SHD as specified in section 5.1. The blue area shows the 95% confidence interval.

In Figure 6 it is clearly visible that multiple runs can highly increase the number of correctly discovered edges.

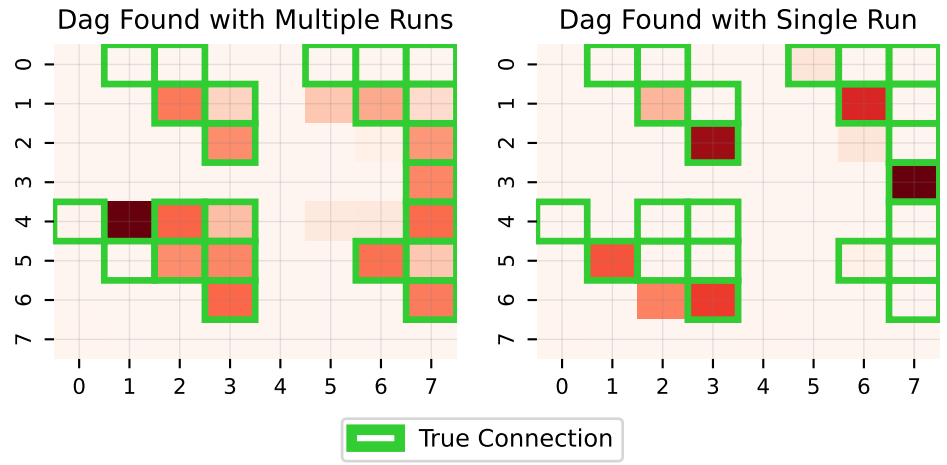


Figure 6: Experiment depicting W^* for a single run of *NOTEARS* (right) and W^* for multiple runs while varying the hyperparameter $l_1 \in \{0, 0.01, 0.1, 1\}$ (left). The true underlying graph is marked with green squares.

5.4 Runtime

The runtime of *NOTEARS* is determined by the matrix exponential which has time complexity in $\Theta(d^3)$. This is only the case if we assume there is a finite number of dual ascent iterations. In the paper, the authors claim that *NOTEARS* approximately has 10 iterations of dual ascent, which my experiments reflect with around 11 iterations. There is no formal prove for this, however, and knowing if there is a scenario where the convergence is not guaranteed might still exist.

5.5 Substructure Recognition

This experiment evaluates the *NOTEARS* algorithm's proficiency in identifying specific substructures *fork* ($X \to Y$ and $X \to Z$), *collider* ($X \to Y \leftarrow Z$), and *chain* ($X \to Y \to Z$) with a focus on the impact of reduced noise in data generation. Reduced noise leads to datasets with more subtle causal signals, making the differentiation between closely related structural equivalence classes more challenging. In these scenarios, the distinctions among potential graphs become less clear, complicating the algorithm's task of accurately inferring the underlying structure.

By constraining the randomness, the aim is to probe the algorithm's sensitivity to the nuanced statistical dependencies that characterise each substructure. This approach highlights *NOTEARS*' potential vulnerabilities, particularly in conditions where causal

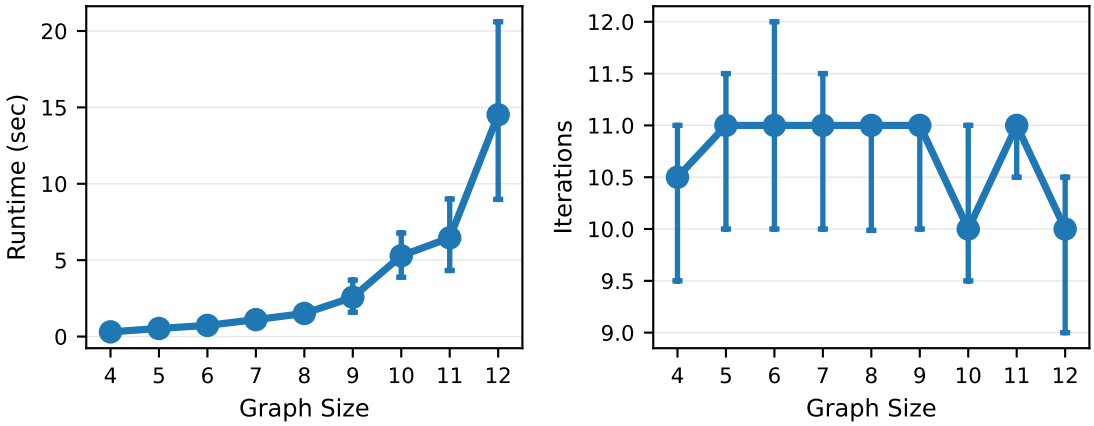


Figure 7: Experiment depicting the time complexity of *NOTEARS* run on simulated data with varying number of variables. On the left the mean runtime in second is shown with 95% confidence intervals (absolute values are meaningless and hardware dependent). On the right plot the median number of iterations of dual ascent until convergence are shown with 95% confidence intervals.

effects are weak and equivalence classes of graphs are densely packed.

Figure 8 depicts the results of the experiment. It is clearly visible that the *NOTEARS* algorithm exhibits significant challenges in accurately detecting the *fork* and *chain* substructures under conditions of reduced randomness. The inherent subtlety in causal signals, resultant from minimised variability, obscures the statistical evidence necessary for detecting the precise nature of these causal relationships. In the case of *fork* structures, where a single cause leads to multiple effects, and *chain* structures, characterised by a sequence of causal links, the dimmed statistical signals fail to provide the clear distinctions *NOTEARS* relies on for accurate structure inference. This results in a pronounced difficulty for the algorithm to identify and correctly model the underlying causal pathways within these configurations.

Conversely, the *collider* structure does not presents challenge for *NOTEARS* in the same experimental setup. The unique statistical signature of colliders—where two independent causes converge on a common effect—remains relatively robust against the effects of reduced randomness. The conditional independence introduced by controlling for the common effect in a *collider* offers a distinctive pattern that *NOTEARS* can exploit more reliably, even when causal signals in the data are subtle. Thus, while reduced randomness diminishes the algorithm's effectiveness in discerning *fork* and *chain* structures due to the blurring of statistical cues, the inherent characteristics of *colliders* facilitate a more consistent identification by maintaining detectable conditional dependencies amidst low variability.

This differential sensitivity underscores the nuanced challenge for *NOTEARS* faces

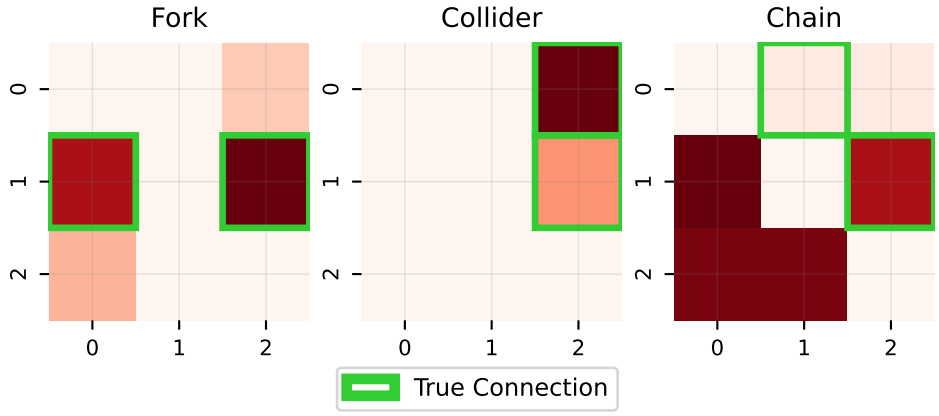


Figure 8: Depiction of an experiment run on data generated from substructures *fork*, *collider* and *chain* with low error variation in the data generation process. The true underlying graph is marked with green squares.

across various structural configurations when operating under constrained randomness.

6 Future Work

6.1 DAG Initialisation

Unfortunately, the authors did not give any notes or tests on how the initial guess for the adjacency matrix ought to be made. Here I want to share some of the findings I made.

One of the first issues in converging to an optimal graph arises from initialising the adjacency matrix as a zero matrix. One can show that for symmetric initialisation of $W^{(0)}_{\alpha^{(0)}}$ the *NOTEARS* algorithm will converge to a DAG with no edges. This happens only under the following condition:

$$\Delta_W F(W, X) = \Delta_W F(W^T, X) \tag{23}$$

i.e. the derivative of the loss function is symmetric with respect to W, which is for instance the case with the linear system of equation loss (see Equation 7). The reason it will converge to a graph with no edges is that the gradient of the Lagrangian $\frac{\mathrm{d}}{\mathrm{d}W_{i,j}} L^\rho(W, \alpha)$ is equal to $\frac{\mathrm{d}}{\mathrm{d}W_{j,i}} L^\rho(W, \alpha)$ which causes all weights to vanish towards 0 in order not to violate the acyclicity constraint. Note that symmetric adjacency matrices are never acyclic.

This small restriction being noted, it should in general be better to initialise the adjacency matrix randomly as the *NOTEARS* algorithm can get stuck in local minima as it is the case with other weight based learning algorithms.[3] Repeated runs of the algorithm with random weight initialisation will therefore lead to more robust findings. Figure 6 is a good illustration of this and further enhances the argument of using random initialisation for $W^{(0)}_{\alpha^{(0)}}$. However,

further research would need to be invested into finding a suitable distribution from which to sample each weight in $W^{(0)}_{\alpha^{(0)}}$.

6.2 Quadratic Penalty Term & Progress Rate

The quadratic penalty term ρ of the *NOTEARS* algorithm is ill defined and using an exponential increase of the variable seems arbitrary (see line 6 in Algorithm 1). An interesting further investigation would be to not use an exponentially increasing function on ρ to satisfy some progress rate but rather an adaptive function that allows the algorithm to be more flexible with respect to the search direction.

7 Conclusion

In conclusion, this seminar report offers a detailed examination of the *NOTEARS* algorithm, showcasing its novel solution to the combinatorial challenges associated with learning Bayesian networks via structural learning. By transforming the issue into a continuous optimization framework, *NOTEARS* facilitates a more efficient and scalable identification of directed acyclic graphs (DAGs) from datasets. The experimental analyses affirm the algorithm's effectiveness and pinpoint areas for future enhancement, particularly regarding algorithm initialization, penalty terms, and progress rates. Additionally, employing model averaging with *NOTEARS* is recommended to achieve more reliable results. Thus, this investigation not only furthers the development of the *NOTEARS* algorithm but also paves the way for potential refinements.

A Graph Properties

In the following rigorous definitions are given for a given directed graph as presented in section 2.1. We define a directed graph G as the ordered pair $G = (V, E)$, where V is a discrete set of vertices and $E \in V \times V$ is the set of directed edges:

(i) **Parents:** The set of parents of a vertex $v \in V$ for a given graph $G = (V, E)$ is defined as

$$\mathrm{Pa}(v) = \{u \in V \mid (u, v) \in E\} \tag{24}$$

In this report represented as a function $\mathrm{Pa} : V \to \mathcal{P}(V)$, where $\mathcal{P}$ denotes the powerset.

(ii) **Ancestors:** The ancestor function gets all parents recursively. We therefore get $\mathrm{Anc} : u \to \mathcal{P}(u)$, with

$$\mathrm{Anc}(u) = \mathrm{Pa}(u) \cup \left(\bigcup_{v \in \mathrm{Pa}(u)} \mathrm{Anc}(v) \right) \tag{25}$$

(iii) **Children:** The set of children of a vertex $v \in V$ for a given graph $G = (V, E)$ is defined as

$$\mathrm{Ch}(v) = \{u \in V \mid (v, u) \in E\} \tag{26}$$

(iv) **Descendants:** The descendant function gets all children recursively. We therefore get $\mathrm{Desc} : V \to \mathcal{P}(V)$, with

$$\mathrm{Desc}(u) = \mathrm{Ch}(u) \cup \left(\bigcup_{v \in \mathrm{Ch}(u)} \mathrm{Desc}(v) \right) \tag{27}$$

(v) **Path:** For a directed graph $G = (V, E)$, a path of length $n \in \mathbb{N}$ in G is a sequence of vertices $S \in V^n$ such that $(S_i, S_{i+1}) \in E$ for $1 \leq i < n$.

(vi) **Undirected Path:** For a directed graph $G = (V, E)$, a path of length $n \in \mathbb{N}$ in G is a sequence of vertices $S \in V^n$ such that $(S_i, S_{i+1}) \in E$, or $(S_{i+1}, S_i) \in E$ for $1 \leq i < n$.

(vii) **Acyclicity:** A graph $G = (V, E)$ is acyclic if there **does not** exist a sequence of distinct vertices $\{v_1, v_2, \ldots, v_n\} \subseteq V$ such that $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)\} \subseteq E$. In other words, there is no closed path in G with distinct vertices, and the graph is devoid of any cycles.

References

[1] Rupesh Agrahari et al. "Applications of Bayesian network models in predicting types of hematological malignancies". In: *Scientific Reports* 8.1 (May 2018). ISSN: 2045-2322. DOI: 10.1038/s41598-018-24758-5.

[2] Steen A. Andersson, David Madigan, and Michael D. Perlman. "A characterization of Markov equivalence classes for acyclic digraphs". In: *The Annals of Statistics* 25.2 (Apr. 1997). ISSN: 0090-5364. DOI: 10.1214/aos/1031833662.

[3] Jaime G. Carbonell. *Neural Networks. Tricks of the Trade*. Ed. by Jörg Siekmann and Gerhard Goos. 1st ed. Description based on publisher supplied metadata and other sources. Berlin/Heidelberg: Springer Berlin Heidelberg, 1998. 1424 pp. ISBN: 9783540494300.

[4] Norman Fenton et al. "Predicting software defects in varying development lifecycles using Bayesian nets". In: *Information and Software Technology* 49.1 (Jan. 2007), pp. 32–43. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2006.09.001.

[5] Charles E. Kahn et al. "Construction of a Bayesian network for mammographic diagnosis of breast cancer". In: *Computers in Biology and Medicine* 27.1 (Jan. 1997), pp. 19–29. ISSN: 0010-4825. DOI: 10.1016/s0010-4825(96)00039-x.

[6] Daphne Koller and Nir Friedman. *Probabilistic graphical models. Principles and techniques*. [Nachdr.] Adaptive computation and machine learning. Includes bibliographical references and index. Cambridge, Mass. [u.a.]: MIT Press, 2010. 1231 pp. ISBN: 9780262013192.

[7] Brady Neal. *Introduction to Causal Inference*. Course Lecture Notes (draft). Prerequisites: Basic probability. Topics from statistics and machine learning are also discussed. 2020.

[8] OEIS Foundation Inc. *Sequence A003024 - Number of acyclic digraphs with n nodes*. [Online; accessed 2024-03-01]. 2020. URL: https://oeis.org/A003024.

[9] Karen Sachs et al. "Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data". In: *Science* 308.5721 (Apr. 2005), pp. 523–529. ISSN: 1095-9203. DOI: 10.1126/science.1105809.

[10] Bin Zhang et al. "Integrated Systems Approach Identifies Genetic Nodes and Networks in Late-Onset Alzheimer's Disease". In: *Cell* 153.3 (Apr. 2013), pp. 707–720. ISSN: 0092-8674. DOI: 10.1016/j.cell.2013.03.030.

[11] Xun Zheng et al. *DAGs with NO TEARS: Continuous Optimization for Structure Learning*. 2018. DOI: 10.48550/ARXIV.1803.01422.

[12] Kai Zhong et al. "Proximal quasi-newton for computationally intensive l1-regularized m-estimators". In: *Advances in Neural Information Processing Systems* 27 (2014).