ALPHA-DAG: A REINFORCEMENT LEARNING BASED ALGORITHM TO LEARN DIRECTED ACYCLIC GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Directed acyclic graphs (DAGs) are widely used to model the casual relationships among random variables in many disciplines. One major class of algorithms for DAGs is called 'search-and-score', which attempts to maximize some goodness-of-fit measure and returns a DAG with the best score. However, most existing methods highly rely on their model assumptions and cannot be applied to the more general real-world problems. This paper proposes a novel Reinforcement-Learning-based searching algorithm, Alpha-DAG, which gradually finds the optimal order to add edges by learning from the historical searching trajectories. At each decision window, the agent adds the edge with the largest scoring improvement to the current graph. The advantage of Alpha-DAG is supported by the numerical comparison against some state-of-the-art competitors in both synthetic and real examples.

1 INTRODUCTION

Directed acyclic graph (DAG) models are widely used to represent directional relations or parent-child relations among interacting units, which has been increasingly studied in diverse disciplines including genetics (Sachs et al., 2005; Zhang et al., 2013), causal inference (Spirtes et al., 2000; Peters et al., 2017), finance (Sanford & Moosa, 2012) and machine learning (Koller & Friedman, 2009). Learning the underlying DAG from an observed data has attracted tremendous attentions in the past few decades and still remains an active research area especially in the situation when no prior information is provided (Chickering et al. (2004); Yuan et al. (2019)).

Most existing DAG algorithms can be categorized into three main classes. The first class uses some local conditional independence criterion to pairwisely test the causal relations (Spirtes et al. (2000); Shimizu et al. (2006); Kalisch & Bühlmann (2007); Peters et al. (2014)) and returns the most likely casual ordering/skeleton based on the observed data. The directed structures can be determined up to the Markov equivalence class (Spirtes et al., 2000) without assuming specific noise assumption. However, this kind of methods are usually computationally inefficient especially when the size of the maximal neighborhood q is large. The multiple testing procedures they use may also lead to some conflicting results (Hyttinen et al., 2014; Spirtes et al., 2000). Another class of algorithms, known as the exact learning methods (Koivisto & Koivisto (2004); Campos & Ji (2011); Van de Geer & Bühlmann (2013)), formulates DAG learning into a regularized framework by adding some acyclicality constraints. One key disadvantage is that the total number of constraints imposed to ensure acyclicality is extremely large which makes the graph reconstructing procedure computationally infeasible (Van de Geer & Bühlmann, 2013; Yuan et al., 2019). Moreover, methods in this class are usually extended from undirected Gaussian graphical model (Kalisch & Bühlmann (2007); Van de Geer & Bühlmann (2013); Ha et al. (2016); Ghoshal & Honorio (2017); Nandy et al. (2018); Yuan et al. (2019); Liu et al. (2019); Li & Zhou (2020)) and focus on the linear case, which is somewhat restrictive and difficult to apply in practice.

The last class is the search-and-score-based algorithms (Chickering, 2003; Nandy et al., 2018; Zheng et al., 2018; Zhu et al., 2020; Heckerman et al., 1995; Geiger & Chickering, 2002; Chickering & Heckerman, 1997), which attempts to optimize some score functions over a feasible DAG space. Yet, as pointed out by Chickering et al. (2004) and (Malone et al., 2004), the optimization problem formulated by these score-based algorithms is usually NP-hard by adding the combinatorial acyclicity constraint. The number of feasible DAGs also superexponentially increases with the number of nodes. A few attempts have been made to approach the global optimality in some special cases (Cussens, 2011; Xiang & Kim, 2013; Cussens et al., 2017), most of which rely on local heuristics to guarantee the acyclicity in the sense that all the directed edges need to be sequentially added. Zheng et al. (2018) formulates the classical score-based optimization problem into a continuous optimization task for linear DAG model and its subsequent work (Zheng et al., 2020) makes an extension to learn sparse nonparametric DAGs by using some nonparametric estimators such as Neural networks. Most recently, Zhu et al. (2020) proposes a Reinforcement-Learning-based method (RL) to search for the DAG with the best score. At learning stage, it dynamically updates the adjacency matrix by maximizing the global score, which is the weighted sum of a predefined score function and a smooth characterization penalty (Zheng et al., 2018). Unfortunately, RL fails to ensure the acyclicity and usually returns a dense DAG with many false edges especially when the total number of nodes is large or the true DAG is sparse.

This paper proposes a novel Reinforcement-Learning-based searching algorithm, named Alpha-DAG. It is motivated by the key idea that the Reinforcement Learning agent with stochastic policy can gradually master the optimal direction to reconstruct the graph based on the value functions learned from the collected searching trajectories. As opposed to some state-of-the-art algorithms, such as GES, which strictly enforces acyclicity at a time, Alpha-DAG gradually finds the optimal order to add edges and avoids generating cycles by learning from the historical failures. At each decision window, the agent chooses to add the edge with the largest scoring improvement or stop the searching process. The proposed algorithm is general in the sense that it can be applied to both linear and nonlinear DAG models. The advantage of Alpha-DAG is supported by the numerical comparison against some state-of-the-art competitors in both synthetic and real examples.

2 Problem Statement

Let $\mathbf{X} \in R^{n\times d}$ be an empirical data matrix consisting of n random vectors $x := [x_1, x_2, \ldots, x_d]^T \in R^d$. Each variable x_i is associated with a node i in a d-node Directed Acyclic Graph (DAG) (Hoyer et al., 2009; Peters et al., 2014). We model x_i by some unknown function of its parents together with an additive random noise n_i:

$$x_i := f_i(x_{pa(i)}) + n_i; i = 1, 2, \ldots, d, \tag{1}$$

where $f_i(\cdot)$ denotes the casual relations, $x_{pa(i)}$ represents a set of variables x_j's such that there exists a directed edge from each $x_j \in x_{pa(i)}$ to x_i in the graph, and the noises n_i's are assumed to be jointly independent. Without adding further constraints on the forms of functions f_i and/or noises n_i, we can assume faithfulness and identify only the Markov equivalence class of the graph (Peters et al., 2014; Spirtes et al., 2000).

Given $\mathbf{X}$, our goal is to infer the underlying DAG $G \in \mathbb{D}$ representing the joint distribution $P(x)$, where $\mathbb{D}$ denotes the (discrete) space of DAGs, $G = (V, \mathbf{B})$, on d nodes. $\mathbf{B} \in \{0, 1\}^{d\times d}$ is the binary adjacency matrix, such that $B(i, j) = 1$ if and only if node $i \in pa(j)$. Therefore, the problem is equivalent to finding the optimal adjacency matrix $\mathbf{B}$, which recovers the true causality between each node i with its parents $pa(i)$ in the graph.

2.1 Score Function

A major class of causal discovery algorithms is to compute the value of the score function $S(G)$ for each feasible G, and then search over the space $\mathbb{D}$ to find the DAG with the best score:

$$\min_G S(G), \text{ subject to } G \in \mathbb{D}$$

In this work, we focus on the BIC score which is known to be not only consistent (Haughton, 1988) but also locally consistent for its decomposability (Chickering, 1996). By introducing a set of functions $\mathcal{F} = \{f_i\}'s$ to model the causal relationships associate with G, we can define the BIC score as follows,

$$S_{BIC}(G) = -2 \log p(\mathbf{X}; \mathcal{F}, \mathbf{B}) + d_G \log n, \tag{2}$$

where $\mathcal{F}$ denotes the causal relationships and d_G denotes the number of edges in the DAG G. Note that the explicit forms of f_i's and the distributions of n_i's are only required to be continuous.

Specifically, the BIC score defined in equation (3) can be modified as

$$S_{BIC_1}(B) = \sum_{i=1}^{d}(n \log(\text{RSS}_i/n)) + \#(\text{edges}) \log n \tag{3}$$

where $\text{RSS}_i = \sum_{k=1}^{n}(x_i^k - \hat{x}_i^k)^2$ denotes the residual sum of squares for the i-th node. x_i^k and $\hat{x}_i^k$ represent the original value and the estimation of the i-th variable in the k-th sample given $\hat{pa}(i)$. G is replaced by $\mathbf{B}$ here since the BIC score only depends on the graph connectivity, or to say the adjacency matrix $\mathbf{B}$ given $\mathbf{X}$ and V. Note that first term in (4) is equivalent to the log-likelihood objective used by GraN-DAG (Lachapelle et al., 2019), and the penalty term controls the number of edges in the graph to prevent over-fitting. By assuming that all the d noises n_i's have equal variance (which is a common assumption made by most DAG studies), we have the BIC score, denoted by $S_{BIC_2}(\mathbf{B})$, to be defined as follows,

$$S_{BIC_2}(\mathbf{B}) = nd \log((\sum_{i=1}^{d} \text{RSS}_i)/(nd)) + \#(\text{edges}) \log n \tag{4}$$

Note that the BIC scores defined in equations (3) and (4) do not require an explicit form of f_i's. f_i's can be either linear or non-linear functions and their identifiabilities are all satisfied under some conditions according to Peter & Bühlmann (2014); Chen et al. (2019); Shimizu et al. (2006; 2011) and Peters et al. (2014).

3 Framework of Alpha-DAG

In this section, we discuss the implementation details of the proposed RL based searching algorithm, Alpha-DAG. In each decision window, the RL agent determines the optimal edge to be added with the highest future return. Different from some state-of-the-art algorithms, such as the GES, which strictly enforces acyclicity at a time, Alpha-DAG resets the searching process whenever the acyclicity is violated. The agent with stochastic policy can gradually master the optimal direction to add edges by learning from the historical failures. Figure 1 in the supplement describes the main architecture of Alpha-DAG with some key components.

3.1 An Markov Decision Process (MDP) formulation

We consider the standard reinforcement learning setting where agent-environment interactions are modeled as a Markov Decision Process (MDP) of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward 'function' $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to R$, a transition kernel $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and a discount factor $\gamma \in [0, 1)$. The *agent* interacts episodically with the *environment* at some discrete time scale, $t \in \mathcal{T} := \{0, 1, 2, ..., T\}$ until the *terminal* time step T is reached. Within each action window t, the driver perceives the state of the environment, described by the feature vector $s_t \in \mathcal{S}$, and on that basis chooses an action $a_t \sim \pi(\cdot|s_t) \in \mathcal{A}$. $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ here denotes a stochastic policy. As a response, the environment produces a numerical reward r_t for each intermediate step. In the context of DAG discovery, we want to introduce the following specifics:

State, denoted as s_t, consists of the observed data $\mathbf{X}$ which is static and the dynamic adjacency matrix $\mathbf{B}_t$, i.e. $s_t := (\mathbf{X}, \mathbf{B}_t)$. $B_t(i, j) = 1$ if an edge from node i to j has been added to the graph until time t. $\mathbf{B_0}$ is a zero matrix for the initial state s_0.

Action, denoted as a_t, represents the action the agent takes at s_t. Executing a_t results in a transition from s_t to s_{t+1} by following the transition dynamics p. The dimension of the whole action space $\mathcal{A}$ is $(d \times d) + 1$. At time t, the agent either adds a directed edge from a node i to another node j by assigning value 1 to $B_t(i, j)$, i.e. $a_t = (i - 1)d + j$, or to keep the adjacency matrix unchanged, i.e. $a_t = 0$. Note that by taking action a_t, the transition from s_t to s_{t+1} is deterministic. An action $a_t > 0$ is feasible if and only if the two selected nodes have not been directly connected before time t, i.e. $B_t(i, j) = 0$ and $B_t(j, i) = 0$. Moreover, we use $v_{a_t}^{(o)}$ and $v_{a_t}^{(d)}$ to represent the origin and destination node of the newly added edge.

Termination, The searching process allows an edge directly from node j to i to be added if j can not be reached by i through a multi-step path in the existing graph. The searching process is terminated whenever a cyclicity happens. To more quickly check this condition in each transition, we store the connectivity information in a $d \times d$ matrix $\mathbf{C}$, where the element $c(i,j)$ in the i-th row, j-th column is a binary variable to inform the connectivity between node i and node j. $c(i,j) = 1$ if there exists a path from i to j. We let $\mathbf{C}_t$ be the connectivity matrix before executing a_t and initialize $\mathbf{C}_0$ as a zero matrix. If a_t assigns 1 to $B_t(i,j)$, we add the product of $C_t(,i)$ and $C_t(j,)$ to $\mathbf{C}_t$ to obtain $\mathbf{C}_{t+1}$, where $C_t(,i)$ and $C_t(j,)$ represent the i-th column and the j-th row of $\mathbf{C}_t$, respectively. If there exists a pair of nodes (i,j) to make $c_{t+1}(i,j) = 1$ when $i \neq j$, the searching process stops. Otherwise, it moves to the $(t+1)$-th decision step. $\mathbf{C}$ is restored to a zero-matrix when the searching process has been terminated. Besides the cyclicity, the process is also terminated when $a_t = 0$. In this case, no more edge can be added to increase the total return.

Reward, r_t measures the change of BIC score after taking action a_t. We let RSS$_{it}$ and $\#(\text{edges})_t$ be the residual sum of squares for the i-th variable at time t and the total number of edges in $G_t = (V, \mathbf{B}_t)$, respectively. If $a_t \neq 0$, without loss of generality, we assume that the agent chooses to add a directed edge from node j to node i at time t following the current policy, r_t can be defined as follows according to (4),

$$\begin{aligned} r_t &= S_{BIC_1}(\mathbf{B}_t) - S_{BIC_1}(\mathbf{B}_{t+1}) \\ &= \sum_{l=1}^{d}(n\log(\text{RSS}_{lt}/n)) - \sum_{l=1}^{d}(n\log(\text{RSS}_{l(t+1)}/n)) + (\#(\text{edges})_t - \#(\text{edges})_{t+1})\log n \\ &= n\log(\text{RSS}_{it}/\text{RSS}_{i(t+1)}) - \log n \end{aligned} \tag{5}$$

By assuming equal noise variances, we have

$$\begin{aligned} r_t &= S_{BIC_2}(\mathbf{B}_t) - S_{BIC_2}(\mathbf{B}_{t+1}) \\ &= nd\log((\sum_{l=1}^{d}\text{RSS}_{lt})/(\sum_{l=1}^{d}\text{RSS}_{l(t+1)})) - \log n \end{aligned} \tag{6}$$

On the other hand, $r_t = 0$ when $a_t = 0$. In particular, we let $r_t = -\Delta$ if a_t results in a cycle. Δ here is a sufficiently large positive number, which is 1000 in this paper. To improve the computation efficiency, we save all the RSS$_{it}$'s in a d-length vector $\mathcal{R}$, and update the i-th entry each time when $a_t > 0$ and $a_t^{(d)} = i$. $\mathcal{R}$ is reset to a zero vector whenever a new searching process begins.

The key advantage of our method is that the computation of only one RSS$_{i(t+1)}$ is required to get the reward r_t. However, all the d RSS$_{it}$' are re-calculated in Zhu & Chen (2019) to obtain either $S_{BIC_1}(\mathbf{B}_t)$ or $S_{BIC_2}(\mathbf{B}_t)$ with all the $d \times d$ elements in $\mathbf{B}_t$ being updated each time.

Policy, $\pi(a|s)$ specifies the probability of taking option a at state s regardless of the time step t. We let $Z^\pi(s,a) := \sum_{t'=t}^{T}\gamma^{t'-t}r_t$ be a random variable of the future return starting from (s_t, a_t) when following policy π, whose expectation is denoted by $Q^\pi(s,a)$. For the MDP defined above, we are interested in finding an optimal policy π^* to add edges, such that $Q^{\pi^*}(s,a) > Q^\pi(s,a)$ holds for any (π, s, a). All the possible optimal policies share the same optimal state-action value function Q^*, which is the unique fixed point of the Bellman optimality operator (Bellman, 1966),

$$Q(s,a) = \mathcal{T}Q(s,a) = E(r + \gamma E_{s'\sim p}[\max_{a'} Q(s',a')] \tag{7}$$

where r) is the reward obtained by taking action a at state s.

4 Model Implementation

4.1 Deep Q-Learning

Based on the Bellman optimality operator provided in (7), Watkins & Dayan (1992) proposes the Q-learning to learn the optimal state-action value function Q^* for control. At each time step, we update $Q(s,a)$ as

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{8}$$

where α is a step size and (s,a,r,s') is a transition. Mnih et al. (2015) combined Q-learning with deep neural network function approximators to propose the Deep-Q-Network (DQN) framework. It is assumed that the Q function is parameterized by a network Θ. At each time step t, DQN performs a stochastic gradient descent to update Θ by minimizing the loss functions $L(\Theta)$,

$$L(\Theta) = E_{s,a\sim\pi}[(y - Q(s,a;\Theta))^2] \tag{9}$$

where $y = E_{s'\sim p}[r + \gamma \max_{a'} Q(s',a';\Theta^-)|s,a]$ and Θ^- denotes the target network, which is a copy of Θ and is synchronized with Θ periodically. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\nabla_\Theta L(\Theta) = E_{s,a\sim\pi;s'\sim p}[(r + \gamma \max_{a'} Q(s',a';\Theta^-) - Q(s,a;\Theta))\nabla_\Theta Q(s,a;\Theta)] \tag{10}$$

In practice, we update the parameter Θ by stochastic gradient descent while the expectation in (10) is estimated by a single sample. In our case, the transition probability p is deterministic after picking an action a. Following the setting of (Mnih et al., 2013), we store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ to build a memory set $D = \{e_1, \ldots, e_N\}$. During the inner loop of the algorithm, the update is employed on a small mini-batch drawn at random from the data pool. By employing the experience replay, the agent executes an action according to an ϵ-greedy policy.

4.2 Estimation of Q-functions

To complete the whole procedure, we need to parameterize the Q function by a well-designed graph autoencoder with input being the node observations $\mathbf{X}$ and the adjacency matrix $\mathbf{B}_t$, output being the Q-function. The architecture should infer the causal graph that best describes the data generating procedure. We first reshape original data sample $\mathbf{X} \in R^{n\times d}$ into $\tilde{\mathbf{X}} = \{\tilde{x}_i\}_{i=1}^d \in R^{d\times n}$ where $\tilde{x}_i \in R^n$ is the vector concatenating all the i-th entries of the vectors in $\{x_l\}_{l=1}^n$. In this case, $\tilde{\mathbf{X}}$ can be treated as the node features, which satisfy the input format of any graph-based neural networks.

Encoder, we use a three-layer Graph Convolution Network (Kipf & Welling, 2016a) to build the encoder structure. Specifically, for the l-th layer, we have

$$\mathbf{H}_t^{(l+1)} = \delta(\tilde{\mathbf{D}}_t^{-\frac{1}{2}}\tilde{\mathbf{B}}_t\tilde{\mathbf{D}}_t^{-\frac{1}{2}}\mathbf{H}_t^{(l)}\mathbf{W}_1^{(l)})$$

at time step t, where $\tilde{\mathbf{B}}_t = \mathbf{B}_t + \mathbf{I}_d$ is the adjacency matrix of the directed graph $G_t = (V, \mathbf{B}_t)$ with added self-connections. $\mathbf{I}_d$ is the identity matrix. $\tilde{D}_t(i,i) = \sum_j \tilde{B}_t(i,j)$ and $\mathbf{W}_1^{(l)}$ is a layer-specific trainable weight matrix. $\delta(\cdot) = \text{ReLU}(\cdot)$ is an element-wise action function. $\mathbf{H}_t^{(l)} \in R^{d\times D_l}$ is the matrix of activations in the l-th layer; $\mathbf{H}_t^{(0)} = \tilde{\mathbf{X}}$. We use $\tilde{\mathbf{H}}_t = \mathbf{H}_t^{(3)} \in R^{d\times D_3}$ to represent the final output of the three-layer GCN encoder.

Decoder, learns the node relational information from the GCN embedding $\tilde{\mathbf{H}}_t$. and output the $d\times d$ $Q(s_t,\cdot)$ functions for all the $d\times d$ potential actions,

$$Q(s_t,\cdot;\Theta) = \tilde{\mathbf{H}}_t\mathbf{W}_2\tilde{\mathbf{H}}_t^T \in R^{d\times d} \tag{11}$$

where $\mathbf{W}_2 \in R^{D_3\times D_3}$ is the weight matrix to learn. Different from the Graph Auto-Encoder proposed by (Kipf & Welling, 2016b), we add $\mathbf{W}_2$ here to make the output asymmetric. $\Theta = \{\mathbf{W}_1^{(1)}, \mathbf{W}_1^{(2)}, \mathbf{W}_1^{(3)}, \mathbf{W}_2\}$ includes all the parameters to be learned associated with the whole graph-based encoder-decoder architecture.

4.3 Exploration

Since the policy learning may highly depend on the initial start and the searching order, we encourage the agent to explore unknown space during the early stage of the training process. We implement the exploration strategy by adding a bonus term to the original Q function when picking the optimal action, which tells the agent whether to try a new direction or continue doing the best thing it knows so far. In this paper, we employ a modified UCB-like strategy (Auer, 2002; Auer et al., 2002) and the optimal action is determined by

$$a = \max_a Q(s,a) + u_t\sigma_s \tag{12}$$

$u_t = c\sqrt{\log t/t}$ is the bonus rate which decays as the number of training epochs t getting large. $\sigma_s \propto \sqrt{1/N(s)}$ with $N(s)$ being the number of times the agent reaches state s, Thus, a state with lower occurrence frequency enjoys a higher exploration bonus.

Algorithm 1 Alpha-DAG algorithm

```
**Given** maximal number of iterations, origin $\epsilon$-greedy rate, target update frequency $M$, $\epsilon$ ,**k**
**Initialize** replay memory $D$ with capacity $N$; policy network $Q_{policy}$ with random weights $\Theta$;
target network $Q_{target}$ with random weights $\widehat{\Theta}$; UCB bonus rate $u_1$
for episode $m = 1, \ldots, M$ do
    Initialise state $s_1 = (\mathbf{X}, \mathbf{B}_1)$, $\mathcal{R}$ and $\mathbf{C}_0$
    for $t = 0, \ldots, T$ do
        if total steps mod k = 0 then
            $\widehat{\Theta} \leftarrow \Theta$
        end if
        With probability $\epsilon$ select a random action $a_t$
        Otherwise, select $a_t = \max_a Q_{policy}(s_t, a; \Theta) + u_m \sigma_s$
        Execute action $a_t$ to compute the reward $r_t$ and obtain the next state $s_{t+1}$
        Update $\mathbf{C}_t$ to get $\mathbf{C}_{t+1}$; Update $\mathcal{R}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathbf{D}$
        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $\mathbf{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal} s_{j+1}; \\ r_j + \gamma \max_{a'} Q_{target}(s_{j+1}, a'; \widehat{\Theta}) & \text{for non-terminal} s_{j+1}. \end{cases}$
        Perform a gradient descent step on $(y_j - Q_{policy}(s_j, a_j; \Theta))^2$ according to equation (10)
    end for
end for
```

5 Numerical analysis

In this section, we examine the empirical performance of Alpha-DAG and some state-of-the-art algorithms on both synthetic and real examples. Note that all the synthetic examples are identifiable and thus the underlying DAG can be exactly reconstructed (Peters et al., 2017). The algorithms being compared include the PC algorithm (Spirtes et al., 2000), LiNGAM (Shimizu et al., 2006), GES (Chickering, 2003; Nandy et al., 2018), the Causal Additive Model (CAM) (Bühlmann et al., 2014), NOTEARS (Zheng et al., 2018), DAG-GNN (Yu et al., 2019), and RL (Zhu & Chen, 2019), whose implementation details and parameter settings are provided in the supplements. For all the compared algorithms, their implementations are conducted by using the default hyper-parameters unless otherwise stated. For pruning, we use the same thresholding method for ICA-LiNGAM, NOTEARS, and DAG-GNN.

Alpha-DAG uses a three-layer GCN with $[500, 1000, 500]$ hidden units as the graph encoder. The target network is updated every 1000 training steps, and the learning rate decays to its 0.99 big after every 1000 training steps. The batch size we use in this paper is 64. For the synthetic examples with small d ($d = 12$), we set γ as 0.8, the learning rate as 1e-4, the maximal iteration time as 15000 and the memory size as 100000. For ϵ-greedy, we initialize ϵ with 0.75 and let it decay to 0.01 linearly within 11000 iterations. The bonus rate of UCB decays to 0 within 12000 iterations. For the case $d = 30$, we set the starting learning rate as $5e-5$, the maximal iteration numbers as 30000 and the memory size as 150000. The initial value of ϵ is still 0.75 but the decay period is set to be 25000. In the last real example, the settings remain the same except γ is set to be 0.3.

For Alpha-DAG, we record all the graphs generated during the training process and output the one with the highest score, which in practice may contain nonexistent edges and further pruning is needed. We calculate the coefficients by fitting all the d models in (1), and delete those edges with regression coefficient smaller than 0.3. On the other hand, since BIC is not sensitive to edge directions, it is much likely to produce causally inverted edges at training stage. Thus, we adopt a greedy approach to flip edges, which helps increase the total reward of the resulting graph. If the flipped edges does not generate a cycle, we keep the flip. With these two postprocessing strategies, the results can be significantly improved.

To evaluate the accuracy of the learned DAGs, we consider three metrics: False Discovery Rate (FDR), True Positive Rate (TPR), and Structural Hamming Distance (SHD). A large value of TPR and a small FDR indicate a good estimate. A smaller SHD implies the closeness of the estimated graph and the true graph.

5.1 Linear model with Gaussian and non-Gaussian noise

We first evaluate the performance of Alpha-DAG and its competitors in the linear case, where $f_i(x_{pa(i)})$ is a linear function of $x_{pa(i)}$ for each $i = 1, ..., d$, i.e.

$$X = \mathbf{B}X + n,$$

with $n = (n_1, ..., n_d)^T$. The following two synthetic examples are considered.

Example 1. (Gaussian noise) We generate the true adjacency matrix $\mathbf{B}$ in the sense that all its upper entries are sampled independently from Bern(p) with Bernoulli parameter p. Then, we reparameterize $\mathbf{B}$ by replacing all the entries of value 1 with a value randomly generated from the interval $[-2, -0.5] \cup [0.5, 2]$. All the noise terms are generated independently from $N(0, 1)$.

Example 2. (Non-Gaussian noise) The generating scheme is the same as that in Example 1 except that all the noise terms are generated randomly from Unif(-1,1), Exp(1) or Gumbel$(0, 1)$.

To test the robustness of Alpha-DAG and its competitors on different graph structures, we consider both sparse and dense directed acyclic graphs by setting $p = 0.2$ ($p = 0.5$ in the supplement). The averaged performance of all the eight algorithms in the the 12 node case are obtained by using a relatively small sample size $n = 256$. For our method, we set the maximal number of iterations to 15000 and use ucb-exploration for BIC1 score (AL1) and ϵ-greedy for BIC2 score (AL2).

The numerical results for Examples 1 and 2 are presented in Table 1. It is evident that Alpha-DAG outperforms its competitors in most scenarios. AL2 is the best performer in terms of all three measures, which almost fully recovers the true graphs. On the other hand, AL1 significantly outperforms RL1. It is also interesting to point out that Alpha-DAG can definitely end up with a DAG while the acyclicity can not be guaranteed by either RL1 or RL2. Note that RL1 and RL2 work well under the dense graph case as illustrated in the supplement but get worse in the sparse graph. As opposed,the performance of Alpha-DAG is consistent in both sparse and dense graph cases, largely due to novel adding edge strategy employed.

Table 1: Empirical results on linear-Gaussian and LiNGAM data models with 12-node prob-0.2.

		AL1	AL2	RL1	RL2	CAM	DAG-GNN	GES	LINGAM	NOTEARS	PC
	FDR	0.23±0.09	0.03±0.05	0.59±0.32	0.04±0.06	0.53±0.30	0.22±0.10	0.18±0.01	0.38±0.05	0.04±0.08	0.35±0.02
LiG	TPR	0.78±0.08	1.00±0.00	0.42±0.36	1.00±0.00	0.46±0.31	0.90±0.05	0.96±0.07	0.72±0.10	0.94±0.11	0.66±0.09
	SHD	3.7±1.50	0.3±0.60	11.0±6.9	0.7±1.2	10.3±7.6	4.3±2.1	3.0±1.0	6.3±2.1	1.7±2.9	7.3±3.2
	FDR	0.17±0.11	0.03±0.07	0.46±0.21	0.02±0.04	0.53±0.21	0.29±0.10	0.21±0.03	0.22±0.08	0.05±0.08	0.31±0.11
LiNG	TPR	0.84±0.09	1.00±0.00	0.55±0.26	1.00±0.00	0.46±0.22	0.84±0.06	0.95±0.08	0.84±0.06	0.92±0.09	0.65±0.07
	SHD	3.0±2.9	0.3±0.8	9.3±4.9	0.3±0.5	9.7±5.6	6.3±3.3	3.7±1.0	4.0±1.8	1.7±2.3	7.7±2.6

We also evaluate the performance of the compared algorithms in the $d = 30$ case. The settings are the same as the low dimensional case except that the upper entries are independently sampled from Bern(0.1) and the maximal number of iterations is set to be 30000. The numerical results for Gaussian and non-Gaussian noise are reported in Table 2. We can see that the outperformance of Alpha-DAG becomes more significant especially in FDR. AL2 achieve a very close performance to the $d = 12$ case while RL2 performs much worse than before. The outstanding performance of Alpha-DAG is largely due to its adding edge strategy. As a comparison, RL prefers to generate large graphs and delete nonexistent edges by postprocessing. However, the pruning process does not work well when d is large or the underlying graph is sparse. A more detailed comparison in terms of the FDR and TPR change during the training process is illustrated in Figures 2 and 3 of the supplement.

Table 2: Empirical results on linear-Gaussian and LiNGAM data models with 30-node.

		AL1	AL2	RL1	RL2	CAM	DAG-GNN	GES	LINGAM	NOTEARS	PC
	FDR	0.28±0.27	0.06±0.08	0.66±0.07	0.36±0.06	0.51±0.10	0.38±0.08	0.22±0.10	0.38±0.11	0.00±0.00	0.35±0.06
LiG	TPR	0.77±0.17	0.98±0.02	0.41±0.04	0.81±0.03	0.67±0.09	0.92±0.08	0.90±0.04	0.47±0.10	0.96±0.04	0.51±0.15
	SHD	18.5±20.5	3.7±3.8	47.3±9.3	21.7±3.1	35.3±15.0	26.7±6.4	11.3±7.6	29.0±9.2	1.7±1.5	27.3±9.0
	FDR	0.43±0.01	0.13±0.12	0.62±0.14	0.32±0.13	0.72±0.06	0.41±0.12	0.17±0.05	0.36±0.10	0.02±0.04	0.33±0.07
LiNG	TPR	0.68±0.13	0.96±0.05	0.48±0.14	0.82±0.06	0.40±0.06	0.89±0.04	0.89±0.06	0.47±0.08	0.96±0.03	0.56±0.10
	SHD	25.3±5.5	8.3±7.2	48.0±23.3	29.0±18.5	53.7±13.6	29.8±11.1	8.2±3.7	29.0±7.5	2.3±2.8	25.0±7.9

5.2 Nonlinear model with quadratic function and Gaussian process

Now, we move from the linear case to the non-linear case. We consider two different formulations of $f_i(x_{pa(i)})$ and the experiment designs are originally introduced in Zhu & Chen (2019).

Example 3. (Quadratic function) We first assume that $f_i(x_{pa(i)})$ is modelled by a quadratic function in which $f_i(x_{pa(i)})$ is linear in both first- and second-order terms of $x_{pa(i)}$. The generating scheme is the same as Example 1 except that each coefficient of the first- and second- order term is randomly generated from the interval $[-1, -0.5] \cup [0.5, 1]$ or 0 with equal probability. We remove the edges whose coefficients are all 0.

The averaged performance of the eight algorithms are evaluated with $n = 500$, $d = 10$ and $p = 0.5$. We fit quadratic regression for any given casual relation to compute the BIC score and set the thresholding value as 0.3 for the proposed pruning strategy. Then, an edge is removed if and only if all the coefficients of the corresponding first- and second order terms are zero after thresholding. As Zhu & Chen (2019) mentions, the pruning method will reduce FDR but have little effect on TPR. Therefore, we only apply the pruning procedure to RL2 and compare it with the other algorithms. The averaged numerical results are reported in Table 4. It is clear that AL2 outperforms all the other competitors in all the scenarios.

Table 3: Empirical results on nonlinear models with 10-node using quadratic function.

	AL2	RL2	CAM	DAG-GNN	GES	LINGAM	NOTEARS	PC
FDR	0.05±0.05	0.11±0.06	0.45±0.04	0.12±0.12	0.35±0.14	0.29±0.06	0.33±0.29	0.07±0.07
TPR	1.00±0.00	0.86±0.13	0.65±0.10	0.24±0.14	0.25±0.03	0.35±0.06	0.30±0.17	0.39±0.14
SHD	1.0±1.0	6.3±5.2	18.3±2.9	17.0±3.0	18.3±3.2	16.3±1.2	5.3±1.5	13.3±2.5

Example 4. (Gaussian process) In this case, f_i's are sampled from a Gaussian process with RBF kernel of bandwidth being one and all the d noises n_i's normally distributed. The data generation method is the same as Zhu & Chen (2019).

The averaged performance of the eight algorithms are evaluated using the $(n, d) = (512, 10)$ combination. The number of true edges is set to be 10. We fit Gaussian process regression for any potential casual relation to compute the BIC score and the averaged numerical results are reported in Table 4. Both our methods outperform the other competitors in all the scenarios.

Table 4: Empirical results on nonlinear models with 10-node using Gaussian process.

	AL1	AL2	RL1	RL2	CAM	DAG-GNN	GES	LINGAM	NOTEARS	PC
FDR	0.21±0.21	0.22±0.23	0.25±0.13	0.52±0.08	0.77±0.09	0.70±0.09	0.53±0.07	0.81±0.10	0.84±0.10	0.54±0.08
TPR	0.93±0.12	0.91±0.15	0.82±0.07	0.48±0.02	1.00±0.00	0.23±0.03	0.66±0.08	0.21±0.08	0.32±0.19	0.57±0.15
SHD	4.0±5.3	4.0±5.3	4.7±4.7	13.0±7.5	34.7±4.0	10.7±5.7	10.3±4.9	14.0±7.5	18.0±8.2	9.7±3.8

According to Examples 1, 2 and 4, we find that Alpha-DAG is less sensitive to the BIC selection than RL and performs more stable for both BIC_1 and BIC_2 cases. This indicates that either AL1 or AL2 would be a better choice than RL in practice if no prior information has been provided.

5.3 Real data

In this part, we evaluate the performance of Alpha-DAG on a real dataset Sachs et al. (2005), which studies a protein signaling network based on expression levels of proteins and phospholipids. Note that this dataset is a well-studied example and is a common benchmark in DAG studies. In this experiment, we generate a sample data with 853 observation, and the true causal relations are modelled by 11 nodes and 17 edges.

We use the same parameter configuration as Example 4 and fit Gaussian process regression to model the causal relationship between different cells. The numerical results of all the competitors are reported in Table 5. Both AL1 and AL2 achieve promising results in reconstructing the directional relations, and are comparable with RL1 and RL2 but outperforms the other six competitors.

Table 5: Empirical results on Sachs data.

	AL1	AL2	CAM	DAG-GNN	GES	LINGAM	NOTEARS	PC	RL1	RL2
Pred Size	10	10	11	15	14	8	20	16	10	10
Correct Eages	7	6	5	4	6	3	6	2	6	7
SHD	11	12	15	24	17	15	19	33	12	11

References

P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

P. Auer, C. Nicolo, and F. Paul. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

R. Bellman. Dynamic programming. *Science*, 153:34–37, 1966.

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

C. Campos and Q. Ji. Efficient structure learning of bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

W. Chen, M. Drton, and Y. Wang. On causal discovery with an equal-variance assumption. *Biometrika*, 106:973–980, 2019.

D Chickering. Learning bayesian networks is np-complete. In *Learning from data*, pp. 121–130. Springer, 1996.

D. Chickering and D. Heckerman. Efficient approximations for the marginal likelihood of bayesian networks with hidden variables. *Machine Learning*, 29:181–2120, 1997.

M Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2003.

M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of bayesian networks is np-hard. *Journal of Machine Learning Research*, **5**:1287–1330, 2004.

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

J. Cussens, D. Haws, and M. Studený. Polyhedral aspects of score equivalence in bayesian network structure learning. *Mathematical Programming*, 164:285–324, 2017.

D. Geiger and D. Chickering. Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. *Annals of Statistics*, 30:1412–1440, 2002.

A. Ghoshal and J. Honorio. Learning identifiable gaussian bayesian networks in polynomial time and sample complexity. *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

M. Ha, W. Sun, and J. Xie. Penpc: A two-step approach to estimate the skeletons of high-dimensional directed acyclic graphs. *Biometrics*, 114:146–155, 2016.

D. Haughton. On the choice of a model to fit data from an exponential family. *The Annals of Statistics*, 16(1):342–355, 1988.

D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20:197–243, 1995.

P. Hoyer, J. Janzing, D.and Mooij, J. Peters, and B. Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in neural information processing systems*, pp. 689–696, 2009.

A. Hyttinen, F. Eberhardt, and M. Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. *In Conference on Uncertainty in Artificial Intelligence*, pp. 523–529, 2014.

M. Kalisch and P. Bühlmann. Estimating high-dimensional directed acyclic graphs with the pc-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

T. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

M. Koivisto and K. Koivisto. Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

S. Lachapelle, P. Brouillard, T. Deleu, and S. Lacoste-Julien. Gradient-based neural dag learning. *arXiv preprint arXiv:1906.02226*, 2019.

H. Li and Q. Zhou. Gaussian dags on network data. *arXiv Preprint, 1905.10848.*, 2020.

J. Liu, W. Sun, and Y. Liu. Joint skeleton estimation of multiple directed acyclic graphs for heterogeneous population. *Biometrics*, 75:36–47, 2019.

M. Malone, K. Kangas, M. Järvisalo, M. Koivisto, and P. Myllymäki. Predicting the hardness of learning bayesian networks. *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pp. 2460–2466, 2004.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

P. Nandy, A. Hauser, and M Maathuis. High-dimensional consistency in score-based and hybrid structure learning. *Annals of Statistics*, 46:3151–3183, 2018.

J. Peter and P. Bühlmann. Identifiability of gaussian structural equation models with equal error variances. *Biometrika*, 101:219–228, 2014.

J. Peters, J. Mooij, D. Janzing, and B. Schölkopf. Causal discovery with continuous additive noise models. *The Journal of Machine Learning Research*, 15(1):2009–2053, 2014.

J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference -Foundations and Learning Algorithms*. MIT Press, 2017.

K. Sachs, O. Perez, D. Pe'er, D. Lauffenburger, and G. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308:523–529, 2005.

A. Sanford and I. Moosa. A bayesian network structure for operational risk modelling in structured finance operations. *Journal of the Operational Research Society*, 63:431–444, 2012.

S. Shimizu, A. Hyvärinen, and A. Kerminen. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7:2003–2030, 2006.

S. Shimizu, T. Inazumi, Y. Sogawa, A. Hyvarinen, Y. Kawahara, T. Washio, P. Hoyer, and K. Bollen. Directlingam: A direct method for learning a linear non-gaussian structural equation model. *Journal of Machine Learning Research*, 12:1225–1248, 2011.

P. Spirtes, C. Glymour, R. Scheines, and D. Heckerman. *Causation, prediction, and search*. MIT press, 2000.

S. Van de Geer and P. Bühlmann. ℓ^0-penalized maximum likelihood for sparse directed acyclic graphs. *Annals of Statistics*, 41:536–567, 2013.

C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

J. Xiang and S. Kim. A* lasso for learning a sparse bayesian network structure for continuous variables. *Advances in Neural Information Processing Systems (NIPS2013)*, pp. 2418–2426, 2013.

Y. Yu, J. Chen, T. Gao, and M. Yu. Dag-gnn: Dag structure learning with graph neural networks. *arXiv preprint arXiv:1904.10098*, 2019.

Y. Yuan, X. Shen, W. Pan, and Z. Wang. Constrained likelihood for reconstructing a directed acyclic gaussian graph. *Biometrika*, 106:109–125, 2019.

B. Zhang, Chris. Gaiteri, and et al. Integrated systems approach identifies genetic nodes and networks in late-onset alzheimer's disease. *Journal of Machine Learning Research*, 153:707–720, 2013.

X. Zheng, B. Aragam, P. Ravikumar, and E. Xing. Dags with no tears: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, pp. 9472–9483, 2018.

X. Zheng, B. Aragam, P. Ravikumar, and E. Xing. Learning sparse nonparametric dags. In *The 24th International Conference on Artificial Intelligence and Statistics*, pp. 3414–3425, 2020.

S. Zhu and Z. Chen. Causal discovery with reinforcement learning. *arXiv preprint arXiv:1906.04477*, 2019.

S. Zhu, I. Ng, and Z. Chen. Causal discovery with reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2020.