PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Research

Cite this article: Darvariu V-A, Hailes S, Musolesi M. 2025 Tree search in DAG space with model-based reinforcement learning for causal discovery. *Proc. R. Soc. A* **481**: 20240450.
https://doi.org/10.1098/rspa.2024.0450

Received: 17 June 2024
Accepted: 8 October 2024

Subject Areas:
computer science

Keywords:
causal discovery, reinforcement learning, tree search, combinatorial optimization, causality

Authors for correspondence:
Victor–Alexandru Darvariu
e-mail: victord@robots.ox.ac.uk
Mirco Musolesi
e-mail: m.musolesi@ucl.ac.uk

† The author is currently affiliated with the Oxford Robotics Institute, Department of Engineering Science, University of Oxford.

Electronic supplementary material is available online at https://doi.org/10.6084/m9.figshare.c.7665971.

THE ROYAL SOCIETY PUBLISHING

Tree search in DAG space with model-based reinforcement learning for causal discovery

Victor-Alexandru Darvariu[1,†], Stephen Hailes[1] and Mirco Musolesi[1,2]

[1]University College London, London, UK
[2]University of Bologna, Bologna, Italy

V-AD, 0000-0001-9250-8175; MM, 0000-0001-9712-4090

Identifying causal structure is central to many fields ranging from strategic decision making to biology and economics. In this work, we propose Causal Discovery Upper Confidence Bound for Trees (CD-UCT), a model-based reinforcement learning (RL) method for causal discovery based on tree search that builds directed acyclic graphs (DAGs) incrementally. We also formalize and prove the correctness of an efficient algorithm for excluding edges that would introduce cycles, which enables deeper discrete search and sampling. The proposed method can be applied broadly to causal Bayesian networks with both discrete and continuous random variables. We conduct a comprehensive evaluation on synthetic and real-world datasets showing that CD-UCT substantially outperforms the state-of-the-art model-free RL technique that operates in DAG space and greedy search, constituting a promising advancement for combinatorial methods.

1. Introduction

Causal graphs are probabilistic graphical models that represent causal dependencies between random variables. They are widely used in many empirical sciences and practical scenarios [1,2] and can enable estimating treatment effects, intervening on variables and answering counterfactual queries [3]. Determining causal structure from observational data is a fundamental task that arises, for instance, when performing randomized controlled trials is impossible or unethical. In particular, score-based

methods seek to find one of the optimal causal graphs with respect to a score function [4] such as the Bayesian information criterion (BIC) [5].

Commonly, the problem is framed as determining a directed acyclic graph (DAG), which eliminates feedback loops. This has been shown to be NP-hard [6], driving decades of research into heuristic or exact methods.

Reinforcement learning (RL) has recently been proposed as a means of navigating this search space, motivated by advances on other NP-hard problems. The appeal of RL for causal discovery stems from its *flexibility* regarding the types of random variables, the functional forms of the relationships between parent and child variables and the diversity of the properties of score functions that can be accommodated. By contrast, many classic causal discovery algorithms impose assumptions; for example, LiNGAM [7] requires continuous random variables characterized by linear relationships and non-Gaussian noise. When the assumptions of a method are violated, this can lead to missing and spurious edges [8]. A notable RL method is RL-BIC [9]: a model-free, continuous approach for one-shot DAG generation. However, its performance on real datasets and scalability on large causal graphs remains limited.

In this article, we propose Causal Discovery Upper Confidence Bound for Trees (CD-UCT), a practical yet rigorous RL method for causal discovery based on *incremental causal graph construction*. Unlike RL-BIC, it is model-based and performs tree search in a discrete space using a granular Markov decision process (MDP) formulation. The core intuition is that access to this model substantially improves the quality of exploration over model-free methods. The advantages of model-based RL have been demonstrated for solving a variety of other problems including the construction of undirected graphs [10]. The proposed method enjoys the flexibility of RL and is applicable across many types of Bayesian networks, data generation models and score functions. Furthermore, it is theoretically grounded: given it is based on UCT, it converges to the optimal action as the number of samples grows to infinity [11], akin to the properties of Greedy Equivalence Search (GES) [12].

The valid action space of the RL agent must exclude edges that would introduce cycles. As the depth and breadth of the tree grow, explicitly checking for cycles becomes prohibitively expensive. To address this key challenge, we propose an incremental algorithm for efficiently tracking cycle-inducing possible edges as the construction progresses. We prove its correctness and empirically show that it results in a speedup of more than an order of magnitude on the largest graphs tested compared with naïve cycle checks.

We evaluate CD-UCT on several real world and synthetic benchmarks showing consistently better performance than the state-of-the-art solution in RL for causal discovery (namely, RL-BIC) and Greedy Search in all settings tested. The flexibility of our method is demonstrated by its compatibility with Bayesian networks with both discrete and continuous random variables using a variety of score functions. We also study the impacts of the simulation budget, graph density and dataset size on algorithm performance. Finally, we analyse the scalability of CD-UCT, which is shown to substantially exceed that of RL-BIC by scaling to graphs with up to $d = 50$ nodes.

2. Background and motivation

In this section, we first review the various classes of methods that have been proposed in the past for the causal discovery problem. Subsequently, we motivate the proposed method by comparing and contrasting it with Greedy Search and RL-BIC, which are the two most closely related techniques.

In doing so, we also introduce key concepts behind the Monte Carlo Tree Search (MCTS) framework, which will later be used in the presentation of our proposed method.

(a) Related work

Combinatorial methods: The goal of combinatorial methods is to find the discrete causal structure that optimizes a given score function. Identifying the optimal DAG (we note that there may be more than one) is known to be NP-hard [6]. As such, exact methods based, for example, on dynamic programming [13] or LP formulations [14] are fairly limited in their ability to scale. Alternatively, the problem can be solved approximately, with some such methods scaling successfully to Bayesian networks with thousands [15] or even millions [16] of variables. Local search methods such as Greedy Search in DAG space [17] or Markov equivalence classes [12,18] have been widely successful. In particular, the GES method, which belongs to the latter category, is proven to find the global optimum in the limit of infinite data. However, this guarantee does not hold in the finite data regime.

Continuous methods: A recent line of work employs continuous optimization, relying on a smooth characterization of acyclicity proposed by [19], which is used as a constraint in a continuous optimization program. Other works build further in this direction by modelling nonlinear relationships with neural networks [20,21]. Such techniques, however, are not guaranteed to return DAGs due to the non-convexity of the objective function [22].

Order-based methods: Another important category of approaches adopts a two-step procedure. First, a causal ordering is determined based on the data. Second, the method seeks the best-scoring graph that is consistent with the ordering. This can lead to a substantially smaller and regular search space [23]; however, errors made in the first step may have a downstream impact on the second. There are several notable approaches based on this idea [23,24], including several recent works based on continuous optimization [25,26].

RL for causal discovery: Introduced by [9], RL-BIC is a score-based continuous method that relies on the acyclicity characterization proposed by [19] in conjunction with RL. Since the score function is used to provide rewards, its differentiability is not required. It features an encoder–decoder architecture trained using the policy gradient for one-shot DAG generation. The best-scoring graph found during training is returned as output. CORL [27] built further in this direction by carrying out the search in the space of orderings. Its action space is formulated as the selection of one of the nodes to add to the ordering. The method relies on an additional intermediate reward that leverages the decomposability of the BIC. The authors showed gains in scalability compared with RL-BIC. Lastly, RCL-OG [28] learns the posterior distribution of orderings (rather than optimizing a single ordering), motivated by the possible unidentifiability of the true causal graph in some settings. Results show that RCL-OG can recover the correct posterior in simplified examples as well as match or exceed the performance of related techniques.

RL for combinatorial optimization: The present article belongs to the emerging area of RL for combinatorial optimization [29,30]. Particularly relevant are those works that consider the construction of graphs as an incremental decision-making problem [31–34]. Our approach is most closely related to [10], which proposes a MCTS method for constructing spatial networks.

(b) Monte Carlo Tree Search and algorithm motivation

(i) Search problems and shortcomings of Greedy Search

Search has been one of the most widely utilized approaches for building intelligent agents since the dawn of AI [35, Chapters 3–4]. Search methods construct a *tree* in which the nodes are states in the MDP. Child nodes correspond to the states obtained by applying a particular action to the state at the parent node, while leaf nodes correspond to terminal states, from which no further actions can be taken. The root of the search tree is the current state. The way in which this tree is expanded and navigated is dictated by the particulars of the search algorithm.

Let us first discuss *Greedy Search*. It creates, at each step, a search tree of depth equal to 1 rooted at the current state, evaluates the objective function for each of the child nodes and picks the action

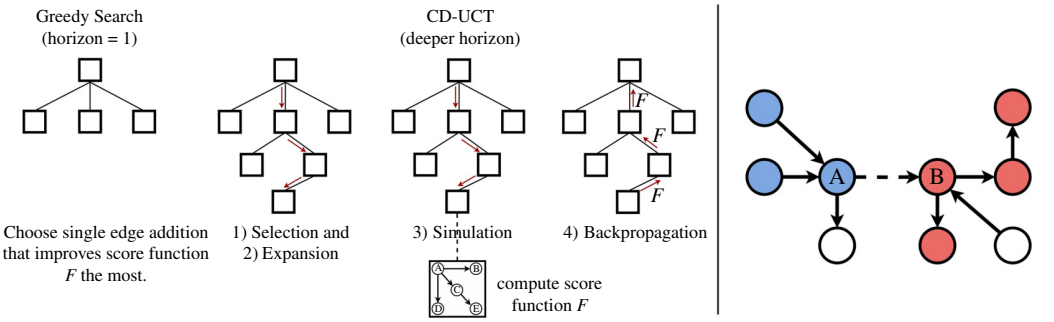

Figure 1. Left: schematic comparison of Greedy Search and CD-UCT, which build shallow and deeper trees, respectively, to search in DAG space. Right: we propose an incremental algorithm to exclude cycle-inducing edges. It relies on the insight that, after adding edge $A \rightarrow B$, connecting a *descendant* (red) of B to an *ancestor* (blue) of A would introduce a cycle in all subsequent timesteps. An illustration of all the algorithm steps can be found in figure 5 in the appendix.

corresponding to the best child node as the next action. The search is repeated with the child node as the root until a terminal state is reached.

Despite its very shallow search horizon, it is commonly used in practice to good effect in a variety of problems [36], GES [12] being a relevant example.[1] The key pathology from which Greedy Search can suffer from is the 'horizon effect' [37], arising due to its shallow search tree. It may make myopic choices as it is unable to explore longer trajectories that contain sets of edges with better scores when considered jointly. Equivalently, actions that are the best locally do not translate to a globally optimal solution. We illustrate this in figure 1.

To see why Greedy Search may be suboptimal, consider the following example in a causal discovery context: two sequences of candidate edges (e_1, e_2) and (e_3, e_4) decrease the BIC score by 1 and 2, respectively, when applied jointly. However, if the decreases for the first edges e_1 and e_3 are 0.5 and 0.1, respectively, a greedy algorithm would not choose the second trajectory due to its myopic horizon.

(ii) Monte Carlo Tree Search

If greedy methods have this shortcoming, why not simply go deeper? In many applications, the branching factor and depth of the search tree make it impossible to explore all paths in the MDP. To break the curse of dimensionality, one option is to use Monte Carlo rollouts: to estimate the goodness of an action, run simulations from a tree node until reaching a terminal state [38,39].

MCTS is an algorithm based on this principle. It is a model-based planning technique that addresses the inability to explore all paths in large MDPs by constructing a policy from the current state [40, Chapter 8.11]. Furthermore, it relies on the idea that the returns obtained by this sampling are informative for deciding the next action at the root of the search tree. We review its basic concepts below, discussing some of the notation used in the pseudocode of algorithm 2. We also refer the interested reader to [41] for a broader review of MCTS.

In MCTS, each node in the search tree stores several statistics such as the sum of returns and the node visit count in addition to the state. For deciding each action, the search task is given a computational budget expressed in terms of node expansions or wall clock time. Specifically, we opt for the former solution and use a budget of simulations per action equal to $b_{\text{sims}} * d$ (hence linearly proportional to the number of nodes in the graph).

At each search step, the algorithm repeatedly executes the procedures below until the search budget is exhausted:

[1]Interestingly, while Christopher Meek's PhD thesis [18], which first proposed GES, acknowledges that Greedy Search is somewhat simplistic and other types of search algorithms should be considered, to the best of our knowledge, the bulk of work has concentrated on GES-like methods.

1. *Selection*: The tree is traversed iteratively from the root node n_{root} until an expandable node (i.e. a node containing a non-terminal state with yet-unexplored actions) is reached.
2. *Expansion*: From the expandable node, a new node is constructed and added to the search tree, with the expandable node as the parent and the child corresponding to a valid action from its associated state. This newly added node is referred to as the *border node* n_{border}. The mechanism for selection and expansion is called *tree policy* (denoted TreePolicy), and it is typically based on the node statistics.
3. *Simulation*: Trajectories in the MDP are sampled from the border node until a terminal state is reached and the return is recorded. The *default policy* or *simulation policy* (denoted SimPolicy) dictates the probability of each action. The simplest version, which we opt for, is to use uniform random sampling of valid actions. We note that the intermediate states encountered when performing this sampling are not added to the search tree.
4. *Backpropagation*: The return is backpropagated from the expanded node upwards to the root of the search tree, and the statistics of each node that was selected by the tree policy are updated.

Once the budget is exhausted, the search step is completed and the action corresponding to the child node with the highest reward (denoted MaxChild) is chosen as the next root node. This process is repeated until a terminal state is encountered. GETACTION and GETSTATE, later used in algorithm 2, are simple helper functions that extract the state and action, respectively, from the search tree node.

The tree policy used by the algorithm trades off exploration and exploitation in order to balance actions that are already known to lead to high returns against yet-unexplored paths in the MDP for which the returns are still to be estimated. The exploration–exploitation trade-off has been widely studied in the multi-armed bandit setting, which may be thought of as a single-state MDP. A representative method is the Upper Confidence Bound (UCB) algorithm [42], which computes confidence intervals for each action and chooses, at each step, the action with the largest upper bound on the reward, embodying the principle of optimism in the face of uncertainty.

Upper Confidence Bounds for Trees (UCT) [11] is a variant of MCTS that applies the principles behind UCB to the tree search setting. Namely, the selection decision at each node is framed as an independent multi-armed bandit problem. At decision time, the *tree policy* of the algorithm selects the child node corresponding to action a that maximizes

$$UCT(s,a) = \bar{r}_a + 2\epsilon_{\text{UCT}}\sqrt{\frac{2\ln C(s)}{C(s,a)}}, \tag{2.1}$$

where $\bar{r}_a$ is the mean reward observed when taking action a in state s, $C(s)$ is the visit count for the parent node, $C(s,a)$ is the number of child visits and ϵ_{UCT} controls the level of exploration.

UCT has been shown to converge to the optimal action with probability 1 as the number of samples grows to infinity [11], and hence features a similar guarantee to GES. However, we note that the number of samples in this context refers to the number of MC simulations that the algorithm is allowed to perform, and not the size n of the considered dataset.

(iii) Model-based versus model-free: why and how CD-UCT outperforms RL-BIC

Next, we provide the reader with an additional comparative analysis of the performance of CD-UCT versus RL-BIC and, more generally, of *model-based* versus *model-free* approaches. Let us first review the distinction between model-based and model-free methods. The former category assumes knowledge of the MDP, while the latter requires only samples of agent–environment interactions. To be specific, in this context, a *model* $\mathcal{M} = (P, R)$ refers to knowing, or having some estimate of, the transition and reward functions P, R. Given our MDP formulation of DAG construction for causal discovery (§3b), in this context, we have access to the 'true' P and R via their mathematical descriptions.

Model-based methods are able to exploit knowledge of $\mathcal{M}$ directly, whereas model-free methods do not. Intuitively, they can determine precisely what the subsequent states and rewards will be for a sequence of actions, without needing to go through a trial-and-error process in the environment, as model-free methods do. Therefore, given an accurate model, model-based algorithms are able to arrive at substantially better policies given the same amount of environment interaction.

Let us give some examples of problems for which the two have been compared and in which results reflect these characteristics. Most relevantly, [10] considered the problem of constructing undirected graphs, finding that a model-based RL method that extends MCTS greatly improves performance and scalability over a variant of the DQN model-free algorithm. [43] showed that UCT outperforms DQN on a range of Atari game environments. The results of [44] highlight that UCT greatly outperforms a REINFORCE model-free agent for the Hex connectionist game.

Returning to the causal discovery problem and the comparative performance of CD-UCT and RL-BIC, as later discussed in the paragraph 'determining budgets' of §4, our experiments are set up such that the budget of score function evaluations awarded to CD-UCT and RL-BIC is the same. The superiority of CD-UCT is manifested as follows: CD-UCT yields substantially better performance when given the same number of score function evaluations (as shown in table 1). Furthermore, figure 2 shows that CD-UCT can match RL-BIC performance with two orders of magnitude fewer score function evaluations, which translates to a DAG of the same quality being found in minutes instead of hours.

Finally, we note that CD-UCT and RL-BIC both store the scores of previously encountered parent sets of each random variable for computing the score function, and thus they are similar in this regard. The two methods differ in how the proposals of possible causal structures are generated, with our granular model-based technique enabling a substantially more efficient navigation of the search space.

3. Methods

(a) Problem formulation

Let $G = (V, E)$ denote a DAG with d nodes and m edges. Each node $v_i \in V$ corresponds to a random variable (RV) x_i that may be discrete or continuous, while edges $e_{i,j}$ indicate directional causal relationships. Let $\mathbf{x}$ denote the vector $(x_1, x_2, \ldots x_d)$ of RVs distributed according to $p(\mathbf{x})$. Let $\mathbf{Pa}(x_i)$ denote the parent set of x_i in the causal graph, i.e. RVs x_k s.t. $e_{k,i} \in E$. Variables x_i are assumed to be independent of their non-descendants given their parent set [4, Chapter 6.5]:

$$p(x_1, \ldots, x_d) = \prod_{i=1}^{d} p(x_i \mid \mathbf{Pa}(x_i)). \tag{3.1}$$

This suffices to represent the problem with discrete RVs. For continuous RVs, we follow the general structural equation model (SEM) of [3], in which RVs are generated according to $x_i = f_i(\mathbf{Pa}(x_i))$, where f_i represents some (typically unknown) function. Many subclasses of SEM exist, with varying assumptions and properties. For example, in the additive noise model [45], variables are generated by applying nonlinear functions to parents and adding arbitrary and jointly independent noise.

Causal discovery: Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ of n d-dimensional observations, the goal is to identify the true underlying DAG G. In *score-based* methods, this is formulated as the optimization of a score function $\mathcal{F}$. Letting $\mathbb{D}^{(d)}$ denote the set of DAGs with d nodes, the problem can be formalized as finding one of the graphs G^* satisfying

$$G^* \in \underset{G \in \mathbb{D}^{(d)}}{\operatorname{argmin}} \mathcal{F}(G). \tag{3.2}$$

Score functions: We consider the BIC, which is known to be consistent and decomposable [46, Chapter 18.3] (noting, however, that the proposed method does not rely on the latter property). For discrete RVs, the BIC $\mathcal{F}_{\mathrm{DV}}$ can be computed as [46]:

$$\mathcal{F}_{\mathrm{DV}}(G) = -\left(\ell\ (G|\mathbf{X}) - \frac{\log n}{2}\sum_{i=1}^{d}\left((\tilde{x}_i - 1) * q_i\right)\right), \tag{3.3}$$

where $\tilde{x}_i$ denotes the cardinality of the discrete RV x_i and $q_i = \prod_{x_j \in \mathbf{Pa}(x_i)} \tilde{x}_j$. The quantity under summation is known as 'network complexity' [47], and it is proportional to the number of free parameters of the factorized joint distribution.

For continuous RVs, following [9], we consider two score functions based on residuals and treat variances as heterogeneous ($\mathcal{F}_{\mathrm{HV}}$) and equal ($\mathcal{F}_{\mathrm{EV}}$):

$$\mathcal{F}_{\mathrm{HV}}(G) = \sum_{i=1}^{d}\left(n \log\left(\mathrm{RSS}_i/n\right)\right) + m \log n \tag{3.4}$$

$$\mathcal{F}_{\mathrm{EV}}(G) = nd \log\left(\left(\sum_{i=1}^{d}\mathrm{RSS}_i\right)/(nd)\right) + m \log n, \tag{3.5}$$

where RSS_i is the residual sum of squares obtained by regressing x_i on its parent variables. Unless otherwise stated, we use Gaussian Process regression [48] for fitting the data and computing the residuals. Intuitively, the search problem can be thought of as finding the DAG that best explains the observed data, with a penalty on the number of edges encouraging sparsity.

We note that, while the term BIC features in the name of the RL-BIC technique, the score functions considered by [9] and used in the implementation of follow-up work [27] are not BIC in a strict sense since the number of parameters that need to be estimated inside the regressor (e.g. pairwise covariances) is not linear in the number of edges. We, nevertheless, choose identical score functions so that the results are directly comparable with those reported in prior works.

(b) DAG construction as MDP

We next describe our formulation of the problem of finding an optimal DAG as an MDP. Unlike RL-BIC, whose graph generation is one-shot, we frame the problem incrementally, i.e. as a decision-making process. We add edges one by one to a (possibly empty) initial graph until an edge budget b is exhausted. To maintain a manageable $\mathcal{O}(d)$ action space, we decompose the addition of an edge into two separate decision-making steps. The MDP components are defined as follows.

State: A state S_t is formed of a tuple $(G_t, \{\sigma_t\})$ containing the DAG $G_t = (V, E_t)$ and a singleton containing an *edge stub* σ_t. At even timesteps ($t \bmod 2 = 0$), σ_t is equal to the empty set $\varnothing$. At odd timesteps, σ_t is equal to v_k, where $v_k \in V$ is the node that was selected in the previous timestep, from which a directed edge must be built.

Action: An action A_t corresponds to the *selection of a node* in V. To ensure that the acyclicity property is maintained, actions that would introduce cycles must be excluded from the action space. Let IsCyclic(G) be a function that, given a DAG G, outputs a Boolean value corresponding to whether the graph contains at least one cycle. We now define the set of edges $\mathcal{C}_t$, whose introduction after time t induces a cycle. Furthermore, let $\mathcal{K}_t(v_i)$ denote those *connectable* nodes v_j such that the candidate edge $e_{i,j}$ is valid:

$$\mathcal{C}_t = \{e_{i,j} \notin E_t \mid \textsc{IsCyclic}(V, E_t \cup \{e_{i,j}\})\}, \tag{3.6}$$

$$\mathcal{K}_t(v_i) = \{v_j \mid\ e_{i,j} \notin \mathcal{C}_t\}. \tag{3.7}$$

The set of available actions containing the nodes that may be selected is defined as below, where $\deg^+(v_i)$ denotes the out-degree of node v_i. The first clause states that maximally connected nodes

or those from which only cycle-inducing edges originate cannot be selected as the edge stub. The second clause, in which v_k denotes the node selected at the previous timestep, forbids actions that would lead to the construction of a cycle-inducing or already-existing edge.

$$\mathcal{A}(S_t) = \begin{cases} \{v_k \in V \quad | \ \deg^+(v_k) < d - 1 \wedge |\mathcal{K}_t(v_k)| > 0\}, \\ \qquad\qquad \text{if } t \bmod 2 = 0 \\ \{v_l \in V \quad | \ e_{k,l} \notin E_t \wedge v_l \in \mathcal{K}_t(v_k)\}, \\ \qquad\qquad \text{otherwise.} \end{cases} \tag{3.8}$$

Transitions: The dynamics is deterministic, meaning that, given a state s and an action a, there is a single state s' that can be reached with probability 1. When transitioning from an even timestep, the model 'marks' the node corresponding to the selected action as the edge stub so that it forms part of the next state. Otherwise, it adds the selected edge to the topology of the next state and resets the edge stub.

Reward: The reward R_t is defined as the negative of the score so as to reconcile the reward maximization paradigm of RL with minimizing the score function. Namely,

$$R_t = \begin{cases} -\mathcal{F}(G_t), \text{ if } t = 2b \\ 0, \text{ otherwise.} \end{cases} \tag{3.9}$$

(c) Incremental algorithm for detecting cycle-inducing edges

Computing the set $\mathcal{C}_t$ (equation 3.6) is required for determining the valid action space (equation 3.8). One strategy is to evaluate the membership condition explicitly for all edges that do not exist in the current edge set (which we refer to as *candidate* edges in the remainder of this work). Cycle existence can be determined by creating a copy of the graph with the candidate edge added and running a cycle detection algorithm (e.g. a traversal such as depth-first search).

However, performing the necessary cycle checks for a set of candidate edges explicitly scales as $\mathcal{O}(d^3)$ using DFS traversals. Given that this must be performed after every edge addition (i.e. every 2 MDP timesteps), this is substantially inefficient, especially if sampling longer trajectories for estimating future reward, as it is typically done in tree search. With this approach, deeper searches are not feasible beyond very small graphs. Instead, we propose a means of keeping track of the cycle-inducing candidate edges by leveraging the fact that our MDP formulation is incremental (i.e. it builds the graph edge by edge).

Let us first introduce some key concepts prior to stating our result. For ease of exposition, we denote a coarser-grained timestep τ, which corresponds to 2 MDP timesteps and advances with each edge addition. We also let $\mathbf{De}(v_i)$ denote the set of *descendants* of v_i (i.e. nodes that can be reached starting from v_i via a directed path); $\mathbf{An}(v_i)$ denote the set of *ancestors* of v_i (nodes that can reach v_i via a directed path). Both sets are taken to be closed (i.e. they include v_i).

Theorem 1. *Let G_τ denote a DAG and known cycle-inducing candidate edges $\mathcal{C}_\tau$. Given that edge $e_{i,j}$ is chosen for addition at timestep τ ($e_{i,j} \in E_{\tau+1}$), the set $\mathcal{C}_{\tau+1}$ is equal to $\mathcal{C}_\tau \cup \Phi_{i,j}$, where $\Phi_{i,j} = \{e_{x,y} \notin E_{\tau+1} \mid (v_x, v_y) \in \mathbf{De}(v_j) \times \mathbf{An}(v_i)\}$.*

The proof is deferred to appendix A. The incremental algorithm making use of this update rule is presented in algorithm 1, and a full run over multiple timesteps is illustrated in the appendix.

A remaining point to address is the initial set $\mathcal{C}_0$. If construction begins from scratch, it trivially holds that the only invalid choice after the addition of edge $e_{i,j}$ is $e_{j,i}$. If starting from an existing graph, $\mathcal{C}_0$ can be computed using traversals or, alternatively, applying the update rule in theorem 1 with an arbitrary ordering of the initial edges. This only needs to be performed *once*

Algorithm 1 Determining cycle-inducing candidates.

Input: timestep τ, DAG $G_\tau = (V, E_\tau)$,
prior cycle-inducing candidates $\mathcal{C}_\tau$,
chosen next edge $e_{i,j}$ to add at time τ.
Output: next cycle-inducing candidates $\mathcal{C}_{\tau+1}$.

if $\tau = 0$ **then** ▷ initialize cycle-inducing candidates
  if $|E_\tau| = 0$ **then** $\mathcal{C}_\tau = \{e_{j,i}\}$
  else
    $\mathcal{C}_\tau = \{\}$
    for $e_{x,y} \notin E_\tau$
      if IsCyclic$(V, E_\tau \cup \{e_{x,y}\})$ **then**
        Add$(\mathcal{C}_\tau, e_{x,y})$
$\Phi_{i,j} = \{e_{x,y} \notin E_{\tau+1} \mid (v_x, v_y) \in \mathbf{De}(v_j) \times \mathbf{An}(v_i)\}$
return $\mathcal{C}_\tau \cup \Phi_{i,j}$

as an initialization step for the search process. We emphasize that, beyond this, *no graph traversals to determine cycles are needed at any point during execution*.

Finally, we would like to note the connection to the concept of *transitive closure* $\mathcal{T}$ of a directed graph, a data structure defined such that $\mathcal{T}_{i,j} = 1$ if there is a directed path from v_i to v_j [49]. For DAG construction, a candidate edge $e_{i,j}$ can be added without introducing a cycle if and only if $\mathcal{T}_{j,i} = 0$. Thus, an alternative approach to algorithm 1 is to compute the transitive closure of the graph and to query it for eliminating cycle-inducing candidates.

There are well-known algorithms (e.g. [50]) for determining the transitive closure of the graph, which may then be updated after every edge addition by incremental procedures [51,52]. However, both algorithm 1 and this alternative approach require time complexity of $\mathcal{O}(d^2)$ since the former computes a Cartesian product while the latter updates and then queries the transitive closure for each candidate. We therefore opt for algorithm 1 due to its simplicity, but note that some speed gains may be possible in practice by pursuing this alternative.

(d) The CD-UCT method

The method, which we term Causal Discovery UCT (CD-UCT), is given in pseudocode in algorithm 2. It builds on the Upper Confidence Bound for Trees (UCT) [11] variant of MCTS. A detailed overview of MCTS was given in §2b, including definitions of the components used in the pseudocode description.

At a high level, the algorithm proceeds in a loop (lines 9–23) to decide a sequence of actions that define a DAG. Each step of the nested loop (14–20) decides an individual action and proceeds until a budget of simulations is exhausted. Within each step, the algorithm navigates the search tree using its tree policy that balances exploration and exploitation and adds a new node (*Selection* and *Expansion*, line 15); samples valid edges using its simulation policy until a terminal state is reached and the reward can be computed (*Simulation*, line 16) and backpropagates the reward to all nodes along the trajectory (*Backpropagation*, line 17). The child of the root node with the highest average reward is chosen as the next root (line 21) and the process repeats. Regarding the differences to standard UCT, we highlight the use of the proposed algorithm 1 to compute the valid action space (10, 11). Furthermore, we note the adoption of the memoization of the best trajectory found during the search (8, 18–20, 22), as in [9,10].

Algorithm 2 Causal Discovery UCT (CD-UCT).

Input: DAG $G_0 = (V, E_0)$, score function $\mathcal{F}$,
edge budget b, simulation multiplier b_{sims},
search horizon h.
Output: actions $A_0, \ldots A_{T-1}$ defining
best-scoring DAG $G_T = (V, E_T)$ w.r.t. $\mathcal{F}$.

$t = 0$, $S_0 = (G_0, \{\varnothing\})$, $r_{\max} = -\infty$
compute $\mathcal{C}_t$ using **algorithm 1**
bestAs = Array(), pastAs = Array()
loop
    update $\mathcal{C}_t$ using **algorithm 1**
    compute $\mathcal{A}(S_t)$ using **equation 3.8**
    if $t = 2b$ **or** $|\mathcal{A}(S_t)| = 0$ **then return** bestAs
    create root node n_{root} from S_t
    for $i = 0$ to $(b_{\text{sims}} * d)$
        n_{border}, treeAs = TreePolicy(n_{root})
        r, outAs = SimPolicy(n_{border}, h)
        Backup(n_{border}, r)
        if $r > r_{\max}$ **then**
            bestAs = [pastAs, treeAs, outAs]
            $r_{\max} = r$
    n_{child} = MaxChild(n_{root})
    Append(pastAs, GetAction(n_{child}))
    $t = t + 1$, S_t = GetState(n_{child})

4. Experiments

(a) Experimental procedure

Datasets with continuous variables: We evaluate the methods on two real-world tasks in the biological domain: *Sachs* [53] and *SynTReN* [54]. The former involves determining causal influences of protein and phospholipid components in the signalling pathways of cells ($d = 11, m = 17$ and $n = 853$). The latter concerns the determination of the structure of gene regulatory networks from gene expression data and consists of 10 transcriptional networks generated by a simulator ($d = 20, m \in [20, \ldots, 25], n = 500$). Both are widely used as benchmarks in the ML for causal discovery literature.

Our focus on realistic data is due to recent work showing that common synthetic benchmarks may be trivial to solve [55]. This is based on the observation that marginal variance in simulated DAGs with the additive noise model tends to increase along the causal order. Hence, a simple baseline that determines a causal order by sorting the variances and performs a sparse regression on the predecessors can be competitive with state-of-the-art algorithms on such synthetic graphs.

Nevertheless, a benefit of synthetic data is that it allows fine-grained control of the generation parameters. With this caveat in mind, we also perform two experiments on graphs with uniformly sampled edges (i.e. Erdős–Rényi). In the first experiment, we consider graphs with $d = 10$ variables generated using Gaussian Process (GP) regression. We study two scenarios: one in which we vary the number of edges $m \in \{15, 20, 25, 30, 35, 40, 45\}$ while holding the dataset size fixed to $n = 10^3$, and another in which we hold $m = 30$ constant and vary the number of datapoints $n \in \{10^1, \ldots, 5 \times 10^3\}$.

For the second experiment, we consider a graph with $d = 50$, $m = 113$ (yielded by an edge probability of 0.1) and $n = 1000$ generated with quadratic regression, which is quicker relative to GP regression for graphs of this scale.

Datasets with discrete variables: We also evaluate applicable methods on classic benchmarks in the Bayesian networks literature: *Asia* [56] ($d = 8, m = 8$), *Child* [57] ($d = 20, m = 25$) and *Insurance* [58] ($d = 27, m = 52$). We use $n = 1000$ samples for each.

Construct-then-prune: We use a two-step procedure for determining the causal graph in the continuous variables case. In the first stage, which we call the *construct* phase, plausible causal relationships are generated. In the second stage, which we call the *pruning* phase, connections are pruned according to some criterion. This paradigm is used by many causal discovery methods [9,12,24]. Following RL-BIC, we use the procedure proposed in CAM by [24] for the pruning phase, which applies a sparse regression [59] and removes edges corresponding to non-significant relationships. For fairness of comparison, all methods undergo CAM pruning. We note that the results of the *construct* phase are still meaningful as they allow us to compare the effectiveness of the search space exploration conducted by the combinatorial methods.

Metrics: We report the reward metric, which directly corresponds to the score and is used by the agents directly as the maximization objective. Additionally, we report the true positive rate (TPR)—the fraction of all causal relationships that are correctly identified; false discovery rate (FDR)—the fraction of causal relationships that are predicted by the model that are incorrect; and structural Hamming distance (SHD)—the minimum number of edge additions, deletions and reversals that are required to transform the output graph into the ground truth causal graph.

Hyperparameters and implementation: We afford CD-UCT and RL-BIC the same number of four hyperparameter configurations tuned on a held-out set of seeds to maximize reward and use the default hyperparameters for the other methods. We use 100 runs for the *Sachs* dataset, 20 each for the *SynTReN* and $d = 10$ synthetic graphs and 50 for the $d = 50$ synthetic graph and the discrete RV graphs. Where applicable, we report means and 95% confidence intervals.[2]

Determining budgets: To ensure fair comparisons between CD-UCT and RL-BIC, we need to align the edge budgets b and score function evaluation budgets b_{sims}. To achieve this, we measure the average number of edges output by RL-BIC in the *construct* phase across multiple runs, yielding $b = 49$ edges for *Sachs* and $b = 97$ for *SynTReN*. Additionally, we set b_{sims} to match the 1.28×10^6 evaluations of the score function performed by RL-BIC with the parameters suggested by the authors. We use $b_{\text{sims}} = 1178$ on *Sachs* and $b_{\text{sims}} = 337$ and $b_{\text{sims}} = 33$ for CD-UCT and Random Search, respectively, on *SynTReN*. The 10× smaller simulation budget for Random Search on the latter dataset is due to its exploration inefficiency leading to longer runtimes (see the 'Runtime analysis' paragraph in the following section). For the synthetic and discrete RV graphs, we set b equal to the true number of edges and $b_{\text{sims}} = 1000$.

(b) Baselines

We compare the proposed method with the following primary baselines that search in DAG space: RL-BIC [9], Greedy Search (GS), Random Search (RS) and Uniform Sampling (US). These methods all operate with an MDP formulation of DAG construction. With the exception of RL-BIC, the MDP defined in §3b is used.

We do not compare directly against the RL-based ordering methods CORL and RCL-OG since they do not operate in DAG space. As we discuss in §2a, searching in ordering space is more tractable at the expense of the potential introduction of errors. Nevertheless, a comparison can be made on the *Sachs* dataset since several works report results on it. CD-UCT achieves a SHD of 10.6 (averaged over 100 runs) by searching in DAG space, while the ordering-based CORL and

[2]Our implementation, data and instructions are available at https://github.com/VictorDarvariu/causal-discovery-mbrl, which enables reproducibility of all the reported results. Further details are provided in appendix B.

RCL-OG yield 13 and 11, respectively, according to single-run results reported in [28]. Therefore, superior results are obtained by CD-UCT on this benchmark despite its operation in a larger search space.

As comparison points, we also consider several non-combinatorial methods that output DAGs. For continuous RVs, we examine the performance of the method against the non-combinatorial methods CAM [24], LiNGAM [7] and NOTEARS [19]. For discrete RVs, we also compare with the exact method GOBNILP [60].

Furthermore, we also consider the GES and Peter–Clark (PC) algorithms. Unlike the approaches listed above, these procedures work in the space of Markov equivalence classes. Therefore, they may return undirected edges as the same conditional independence relations can be encoded by different graph structures [61]. This raises the issue of how the metrics should be adapted to deal with undirected edges. We treat GES and PC favourably by classifying an undirected relationship as a true positive if predicted correctly in either direction. Consequently, we use an asterisk in the relevant tables to indicate that the results for GES and PC are not directly comparable and additionally report the percentage of undirected edges that they output.

We now provide a high-level overview of each of the baseline methods aside from RL-BIC, which was discussed in detail in §2a.

- *Greedy Search (GS)*: This method creates, at each step, a search tree of depth 1 that enumerates all graphs that can be reached via the addition of a single edge. The edge that leads to the largest improvement in the score function is selected for addition, and the process is repeated starting from the resulting graph until the edge budget is exhausted.
- *Random Search (RS)*: This method samples valid actions uniformly at random starting with an empty DAG until a terminal state is encountered and the resulting DAG is scored. The best-scoring DAG found across all performed simulations is output as the result. Since it is given a budget of simulations comparable with CD-UCT and RL-BIC, it gauges the effectiveness with which the search space is navigated.
- *Uniform Sampling (US)*: This strategy corresponds to choosing a graph uniformly at random out of the state space. It is not intended to be competitive and only meant as a comparison point given the metrics are on different scales.
- *Causal Additive Model (CAM)*: CAM [24] is an order-based method that assumes the additive noise model. In the first stage, an ordering of the causal variables is determined using maximum likelihood estimation. In the second stage, sparse regression [59] is used to select edges that are consistent with the ordering.
- *Linear Non-Gaussian Acyclic Model (LiNGAM)*: LiNGAM [7] assumes a linear model with noise generated according to a non-Gaussian distribution. The technique is based on Independent Component Analysis, which is used to obtain a 'mixing matrix'. After appropriate permutation and normalization, it is used to estimate pairwise 'connection strengths' that determine edge existence.
- *Non-combinatorial Optimization via Trace Exponential and Augmented Lagrangian for Structure Learning (NOTEARS)*: NOTEARS [19], also discussed in the main text, proposes to replace the discrete acyclicity constraint with a continuous one. In this way, the causal identification problem is cast as a mathematical program, for which continuous optimization techniques can be used. It assumes a linear structural equation model and learns a matrix of pairwise weights that is thresholded to determine the relationships.
- *GOBNILP*: GOBNILP [60] is an exact method for learning a Bayesian network structure. It is based on a branch-and-cut approach that features an integer programming formulation together with cutting planes that are specifically designed for this problem. Given the practical limitation of needing to find a solution within reasonable computational time, it operates in a best-effort fashion. For challenging problem instances, it therefore may return the best-found graph and an upper bound on the optimal score. Furthermore, to reduce the search space, GOBNILP allows specifying a limit on the size of the parent sets. In our experiments, we set this parameter to 4, which enables GOBNILP to find the optimal

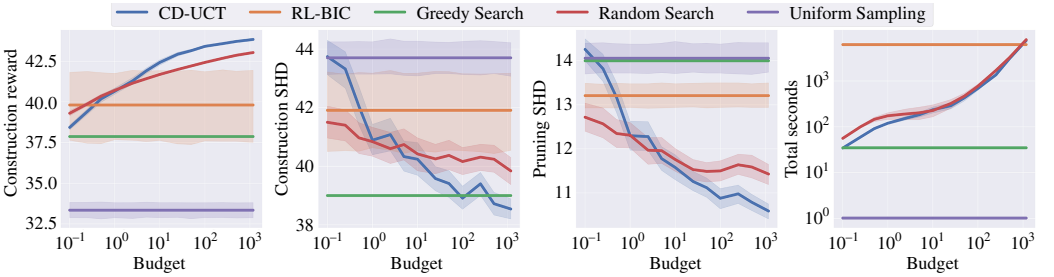

Figure 2. Varying the b_{sims} simulation budget parameter on *Sachs*. The subfigures show the construction reward, structural Hamming distance (SHD) for construction and pruning and wall clock time. CD-UCT and Random Search both outperform RL-BIC, even when given 100 times fewer score function evaluations.

structure for the considered discrete variable datasets, whose RVs have at most three parents. Nevertheless, this requires knowledge about the ground truth graph that is not required by (and supplied to) the other techniques.

- *Greedy Equivalence Search (GES)*: The GES algorithm searches greedily in the space of Markov equivalence classes; we note that it differs from the Greedy Search procedure described above, which searches greedily in the space of directed edge additions instead. The algorithm proceeds in two phases that perform edge additions and deletions, respectively, until local maxima are encountered. It is guaranteed to find the optimal equivalence class in the limit of infinite samples [12]. Typically, however, causal discovery is performed in a finite sample regime in which this guarantee does not hold.
- *Peter-Clark (PC)*: The PC [62] algorithm is a constraint-based method that proceeds in two stages. First, conditional independence tests are carried out in order to determine an undirected graph. Second, the edges are (partially) oriented by performing d-separation tests. Given the method does not prescribe particular tests, it is applicable to a variety of causal discovery scenarios, including time series.

5. Results

In this section, we present results with the $\mathcal{F}_{\text{DV}}$ score function for discrete variables and the $\mathcal{F}_{\text{HV}}$ score function for continuous RVs. For completeness, we repeat the main experiments with the $\mathcal{F}_{\text{EV}}$ score function with the more restrictive 'equal variances' assumption. These results are shown in appendix C and display similar characteristics.

Main results with continuous variables: The key results on the *Sachs* and *SynTReN* datasets are shown in table 1. In the construction phase, CD-UCT performs best out of the considered combinatorial methods in terms of the rewards received and the metrics. The performance improvements with respect to RL-BIC are substantial. Indeed, RL-BIC achieves worse results than a Random Search that is afforded a similar number of score function evaluations.

After pruning, CD-UCT maintains the best results out of all combinatorial methods. It performs the best overall on *Sachs* yielding an SHD of 10.6, but it is outperformed on average by the order-based CAM method on the *SynTReN* graphs. Note that the SHD of RL-BIC on *SynTReN* is lower due to the pruning eliminating more non-significant edges rather than its identification of true relationships, as reflected in the much poorer TPR and FDR. Furthermore, the × in the NOTEARS column is due to the method not returning DAGs for some of the instances in this dataset, as it may become stuck in local optima [22]. GES and PC perform well on both datasets but generally output large fractions of undirected edges.

Budget analysis: We conduct an experiment in which we vary the simulation budget multiplier afforded to the CD-UCT and Random Search agents for the *Sachs* dataset. These results are shown in figure 2, in which the largest value on the x-axis corresponds to equal simulations to RL-BIC.

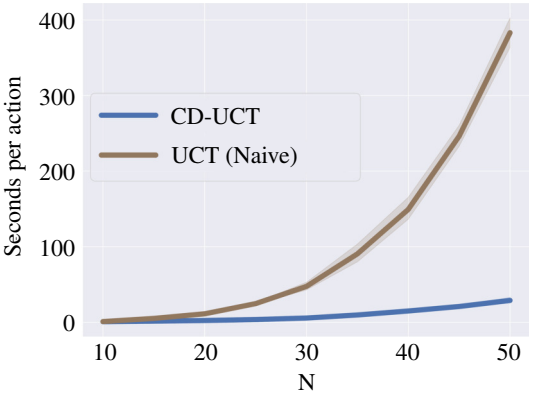

Figure 3. Runtimes of CD-UCT with the incremental algorithm 1 against a naïve implementation that performs traversals to detect cycles.

Beyond a tiny simulation budget, CD-UCT is indeed able to navigate the search space better and find higher-scoring graphs than Random Search. Strikingly, even with a modest simulation multiplier corresponding to approximately 100 times fewer simulations than RL-BIC, both CD-UCT and Random Search yield better reward and SHD. This provides evidence of the advantages of CD-UCT compared with RL-BIC (and, more generally, of model-based approaches compared with model-free ones, as discussed in §2b).

Main results with discrete variables: As shown in table 2, GOBNILP performs best overall, which is to be expected given its exact nature. CD-UCT outperforms the directly comparable combinatorial methods that use the same MDP formulation (GS, RS, US). The performance difference between CD-UCT and GS is smaller relative to the continuous RV case. GES strongly outperforms CD-UCT and GS, which highlights the possible advantages of searching in the space of equivalence classes. The relationship between RS and GS is reversed relative to the continuous RV case: the latter obtains substantially better results for the larger two graphs. This suggests that this setting favours shorter search horizons. The exclusion of RL-BIC from this experiment is due to its incompatibility with discrete RVs. PC only outputs undirected edges for the considered datasets. The missing values for PC on the *Insurance* dataset stem from the inability of the implementation we used to handle datasets with only one observed value for certain RVs.

Impact of incremental algorithm: We examine the impact of algorithm 1 for eliminating cycle-inducing candidates by comparing CD-UCT with a naïve version that performs traversals, all other aspects being equal. This is carried out on graphs with $d \in \{10, 20, 30, 40, 50\}$, edge probability of 0.1, quadratic regression and $b_{\text{sims}} = 10$. We showcase the results of this analysis in figure 3. The speedup scales super-linearly and improves from a factor of approximately 1.25× for graphs with $d = 10$ to 13.25× for graphs with $d = 50$. We conclude that this algorithm is a key component for tree search in DAG space with larger graphs.

Synthetic graph experiments: In figure 4, we examine the impact of the true graph density and dataset size on performance. When the edge count is low, CD-UCT and Greedy Search perform similarly. As the density grows, the two curves diverge: CD-UCT improves substantially (as does Random Search), while Greedy Search becomes akin to uniformly randomly sampling a trajectory. To understand why this occurs, recall that Greedy Search computes the improvement in the score after each edge addition, while CD-UCT and Random Search compute the score at the terminal state of the MDP once the entire hypothesized causal graph has been generated. In sparser graphs, it is common for variables to have few or even no parents, and therefore, the greedy criterion is an effective approximation. In denser graphs, in which parent sets have larger cardinality, this leads to a degradation in performance as Greedy Search can only consider the score contribution

Table 1. Results obtained by the methods in the construction phase (top) and after undergoing pruning (bottom). CD-UCT performs best out of all comparable methods for construction, both in terms of rewards and downstream metrics. Post-pruning, CD-UCT performs best on *Sachs* and is competitive with the other methods on *SynTReN*. GES outputs 75% undirected edges for *Sachs* and 8% for *SynTReN*, while the figures for PC are 75% and 53%.

phase	dataset	metric	CD-UCT	RL-BIC	GS	RS	US	CAM	LiNGAM	NOTEARS	GES*	PC*
construct	*Sachs*	reward ↑	$43.834_{\pm 0.044}$	$39.807_{\pm 2.209}$	37.867	$43.030_{\pm 0.051}$	$33.343_{\pm 0.486}$	—	—	—	—	—
		TPR ↑	$0.664_{\pm 0.013}$	$0.548_{\pm 0.032}$	0.588	$0.620_{\pm 0.022}$	$0.421_{\pm 0.030}$	—	—	—	—	—
		FDR ↓	$0.770_{\pm 0.005}$	$0.811_{\pm 0.008}$	0.796	$0.785_{\pm 0.008}$	$0.854_{\pm 0.010}$	—	—	—	—	—
		SHD ↓	$38.680_{\pm 0.342}$	$41.910_{\pm 1.383}$	39.000	$39.840_{\pm 0.478}$	$43.700_{\pm 0.587}$	—	—	—	—	—
	SynTReN	reward ↑	$97.847_{\pm 0.339}$	$73.874_{\pm 4.348}$	97.700	$89.181_{\pm 0.311}$	$74.717_{\pm 0.676}$	—	—	—	—	—
		TPR ↑	$0.376_{\pm 0.021}$	$0.308_{\pm 0.033}$	0.292	$0.285_{\pm 0.016}$	$0.256_{\pm 0.014}$	—	—	—	—	—
		FDR ↓	$0.907_{\pm 0.006}$	$0.932_{\pm 0.005}$	0.925	$0.930_{\pm 0.004}$	$0.938_{\pm 0.004}$	—	—	—	—	—
		SHD ↓	$96.035_{\pm 1.181}$	$107.387_{\pm 6.930}$	98.000	$100.884_{\pm 0.877}$	$102.155_{\pm 0.899}$	—	—	—	—	—
prune	*Sachs*	TPR ↑	$0.432_{\pm 0.011}$	$0.309_{\pm 0.019}$	0.235	$0.373_{\pm 0.014}$	$0.211_{\pm 0.021}$	0.353	0.235	0.118	0.353	0.353
		FDR ↓	$0.258_{\pm 0.018}$	$0.447_{\pm 0.028}$	0.556	$0.319_{\pm 0.025}$	$0.577_{\pm 0.039}$	0.400	0.556	0.500	0.250	0.250
		SHD ↓	$10.600_{\pm 0.183}$	$13.210_{\pm 0.284}$	14.000	$11.430_{\pm 0.251}$	$14.060_{\pm 0.352}$	12.000	14.000	15.000	11.000	11.000
	SynTReN	TPR ↑	$0.305_{\pm 0.018}$	$0.149_{\pm 0.014}$	0.264	$0.231_{\pm 0.014}$	$0.189_{\pm 0.012}$	0.329	0.303	×	0.460	0.337
		FDR ↓	$0.812_{\pm 0.013}$	$0.887_{\pm 0.010}$	0.839	$0.865_{\pm 0.009}$	$0.879_{\pm 0.008}$	0.794	0.809	×	0.832	0.749
		SHD ↓	$43.890_{\pm 1.158}$	$43.683_{\pm 1.152}$	46.100	$49.352_{\pm 0.990}$	$47.715_{\pm 0.946}$	41.300	41.600	×	61.700	37.800

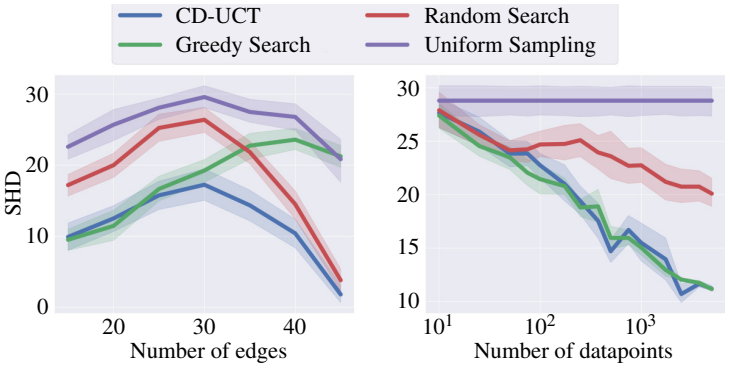

Figure 4. Results with problem instances of varying graph density and number of datapoints.

Table 2. Results with discrete RVs. GES outputs 20%, 8% and 4% undirected edges for the three datasets, respectively, while PC outputs 100% undirected edges (i.e. no directionality can be determined).

dataset	metric	CD-UCT	GS	RS	US	GOBNILP	GES*	PC*
Asia	reward ↑	$-2.171_{\pm 0.002}$	−2.177	$-2.207_{\pm 0.003}$	$-2.727_{\pm 0.044}$	−2.165	—	—
	TPR ↑	$0.505_{\pm 0.062}$	0.250	$0.362_{\pm 0.052}$	$0.128_{\pm 0.033}$	0.750	0.875	0.625
	FDR ↓	$0.495_{\pm 0.062}$	0.750	$0.637_{\pm 0.052}$	$0.873_{\pm 0.033}$	0.250	0.150	0.000
	SHD ↓	$5.120_{\pm 0.503}$	7.000	$7.620_{\pm 0.520}$	$12.620_{\pm 0.571}$	3.000	3.000	3.000
Child	reward ↑	$-13.036_{\pm 0.026}$	−13.018	$-15.154_{\pm 0.041}$	$-17.646_{\pm 0.306}$	−12.972	—	—
	TPR ↑	$0.670_{\pm 0.029}$	0.640	$0.174_{\pm 0.018}$	$0.064_{\pm 0.016}$	0.800	0.720	0.440
	FDR ↓	$0.233_{\pm 0.037}$	0.360	$0.826_{\pm 0.018}$	$0.936_{\pm 0.016}$	0.091	0.514	0.000
	SHD ↓	$9.580_{\pm 1.133}$	12.000	$37.700_{\pm 0.699}$	$45.300_{\pm 0.827}$	5.000	20.000	14.000
Insurance	reward ↑	$-14.740_{\pm 0.029}$	−14.746	$-19.115_{\pm 0.068}$	$-27.093_{\pm 2.220}$	−14.358	—	×
	TPR ↑	$0.331_{\pm 0.014}$	0.308	$0.126_{\pm 0.012}$	$0.078_{\pm 0.010}$	0.481	0.596	×
	FDR ↓	$0.596_{\pm 0.038}$	0.692	$0.874_{\pm 0.012}$	$0.922_{\pm 0.010}$	0.265	0.426	×
	SHD ↓	$54.460_{\pm 2.646}$	57.000	$85.080_{\pm 1.167}$	$92.080_{\pm 1.080}$	28.000	40.000	×

Table 3. Approximate wall clock running time in hours, minutes and seconds with the $\mathcal{F}_{\text{HV}}$ score function.

	CD-UCT	RL-BIC	GS	RS	US	CAM	LiNGAM	NOTEARS	GES	PC
Sachs	1:53:24	1:43:33	0:01:48	3:46:56	0:00:04	0:00:39	0:00:04	0:00:05	0:00:01	0:00:03
SynTReN	2:10:37	35:32:20	0:03:11	10:35:48	0:00:07	0:01:38	0:00:22	0:01:58	0:00:01	0:00:04

of a single edge, and not of a larger set of parent variables. Regarding dataset size, performance improves for all methods as the number of datapoints increases. The performance of Random Search plateaus while the relative ones of CD-UCT and Greedy Search remain similar, which is consistent with the middle-point on the x-axis of the left plot. The number of edges appears to be a stronger determinant for the relative performance of CD-UCT and Greedy Search on these synthetic instances. Finally, there is no definitive winner among Greedy Search and Random Search since each achieves better performance in some settings, as also observed on the real-world benchmarks.

Runtime analysis: In table 3, we show the wall clock runtimes associated with the results presented in table 1. CD-UCT yields similar runtimes to RL-BIC on the *Sachs* dataset, but it

Table 4. Results with $d = 50$ synthetic continuous RVs. GES and PC output 1% and 8% undirected edges, respectively.

	CD-UCT	RL-BIC	GS	RS	US	CAM	LiNGAM	NOTEARS	GES*	PC*
Reward ↑	$303.466_{\pm 2.532}$	∞	275.519	$81.903_{\pm 1.281}$	$14.270_{\pm 2.490}$	—	—	—	—	—
TPR ↑	$0.852_{\pm 0.009}$	∞	0.841	$0.100_{\pm 0.006}$	$0.046_{\pm 0.006}$	0.611	0.381	0.124	0.442	0.177
FDR ↓	$0.148_{\pm 0.009}$	∞	0.159	$0.900_{\pm 0.006}$	$0.954_{\pm 0.006}$	0.623	0.865	0.500	0.780	0.767
SHD ↓	$30.880_{\pm 1.704}$	∞	34.000	$198.020_{\pm 1.243}$	$210.220_{\pm 1.353}$	149.000	326.000	107.000	227.000	141.000

is more than an order of magnitude faster on *SynTReN* despite not being implemented in a low-level programming language. This is a consequence of the targeted exploration of the search space, which leverages the decomposability of BIC to avoid regressions for already-seen parent sets of a given variable. Random Search displays the same inefficiency in terms of exploration and requires 5× more time despite performing 10× less simulations than CD-UCT on the *SynTReN* dataset. The methods that do not perform the same number of score function evaluations or are based on a different paradigm altogether are characterized by a lower wall clock time, partially due to implementation concerns. In particular, the GES implementation that we interface with is very fast due to its use of C++ bindings. The runtime of a pure Python implementation of GES would normally exceed that of Greedy Search, since GES also performs an edge deletion phase and requires tracking equivalence classes. Finally, we note that the runtimes of the other methods that construct graphs incrementally (Greedy Search, Random Search, Uniform Sampling) also benefit from the incremental algorithm for determining cycle-inducing edges, as they share the same environment for graph construction.

Scaling to larger graphs: Our final experiment is performed with a synthetic graph with $d = 50$ as previously described. These results are displayed in table 4. CD-UCT shows statistically significantly better performance in terms of reward and metrics with respect to Greedy Search, albeit only marginally for the latter. RL-BIC cannot function with graphs of this size as it needs to model d^2 edge probabilities, which is unsurprising given the remarkable increase in wall clock time observed from $d = 11$ to $d = 20$ as shown in the paragraph above. We attribute the better performance of score-based methods relative to CAM, LiNGAM, NOTEARS, GES and PC to the use of quadratic regression both in the ground truth generation and the scoring function, acting as a strong model class prior.

6. Conclusion

Summary: In this work, we have proposed CD-UCT, a practical yet rigorous method for causal structure discovery based on an incremental construction process through RL. The latter is an attractive paradigm due to its flexibility regarding the types of random variables, data generation models and score functions it can accommodate. We have also derived an algorithm for efficiently determining cycle-inducing edges as construction progresses, which substantially improves running times. We have demonstrated significantly better performance than RL-BIC and Greedy Search on two real-world tasks, while showing that RL-BIC often performs worse than a Random Search in DAG space. Furthermore, we have examined the impact of graph density and dataset size on the relative performance of the methods and shown that, unlike RL-BIC, our method can operate on graphs with $d = 50$ nodes.

Implications: Our results highlight that model-based RL can substantially outperform model-free RL techniques for causal discovery, echoing findings in other applications for which they have been compared. Additionally, we also showed that the myopic horizon of Greedy Search may be suboptimal, and a deeper search can yield better causal structures in a variety of settings.

Given that greedy approaches are a component in many algorithms—e.g. they are used to search the space of equivalence classes in GES [12] and the space of orderings in CAM [24]—performing a deeper search may find broader uses with different state and action space formulations. Our results have also shown that CD-UCT can attain competitive performance on widely accepted benchmarks and problem settings despite its lack of problem-specific assumptions, which stresses the potential of the RL framework for approaching causal discovery. Other settings in which the assumptions of many classic methods would be violated, whereas ours is soundly applicable, include score functions that are not decomposable (e.g. the generalized score functions proposed by [8]), problems with mixed discrete and continuous random variables and problems with diverse types of statistical relations (e.g. both linear and nonlinear dependencies within the same causal discovery instance). We leave the application of CD-UCT in such scenarios to future work.

Extensions and improvements: CD-UCT can be adapted to search in ordering space by modifying the MDP definition. This would involve episodes of length d to construct an ordering followed by a post-processing step to obtain a DAG. We leave this step, as well as experimental comparisons with other ordering-based techniques, for future investigations. A highly practical improvement for reducing the DAG search space and improving the scalability of CD-UCT is to impose a limit on the sizes of parent sets of the RVs, as performed by GOBNILP. This can be realized by modifying the second clause of equation 3.8 to include an additional constraint on the in-degree of node v_l. Another interesting research direction is the improvement of the simulation policy at the heart of our approach. Currently, it treats each edge as equally likely when performing rollouts, which is free from bias but may result in noisy estimates of future rewards. Prioritizing edges according to statistical relationships present in the dataset or external knowledge provided by an expert may improve the accuracy of the discovered structures. Finally, as it is based on MCTS, the method can be extended to use neural networks to bias the navigation of the search space. The use of policy and value networks, as leveraged by AlphaGo [63] and its successors, have proven successful in a variety of domains. However, a challenge that differentiates causal discovery is the uniqueness of each instance of the problem, as the meaning of the measurements captured by the random variables differs between datasets. Therefore, for function approximation techniques to offer meaningful generalizations, they must be applied to causal discovery instances that are closely related.

Other applications: The proposed RL method can be applied beyond causal discovery, since it does not rely on assumptions about the state space and reward functions, as long as they consist of and can be applied to DAGs. As such, the method is directly applicable to other scenarios where finding DAGs that optimize an objective function is relevant. Notably, DAGs are often used to represent *dependencies*, e.g. those between tasks that must be scheduled in time-shared concurrent computer systems [64], relationships between software packages [65] or components in a manufacturing pipeline [66].

Data accessibility. Our experiments use benchmark datasets that are publicly available without licensing restrictions. They are included in the electronic supplementary material in the appropriate formats [69].

Declaration of AI use. We have not used AI-assisted technologies in creating this article.

Authors' contributions. V.A.D. conceptualization, data curation, investigation, methodology, project administration, software, validation, visualization, writing—original draft, writing—review & editing; S.H: conceptualization, funding acquisition, methodology, supervision, writing—review & editing; M.M: conceptualization, funding acquisition, methodology, supervision, writing—review & editing.

All authors gave final approval for publication and agreed to be held accountable for the work performed herein.

Conflict of interest declaration. We declare we have no competing interests.

Funding. This work was supported by the Turing's Defence and Security programme through a partnership with the UK government in accordance with the framework agreement between GCHQ & The Alan Turing Institute.

Acknowledgements. Experiments for this article were carried out on the UCL Department of Computer Science High Performance Computing (HPC) cluster.

Appendix A. Proof of theorem 1

Theorem 1. *Let G_τ denote a DAG and known cycle-inducing candidate edges $\mathcal{C}_\tau$. Given that edge $e_{i,j}$ is chosen for addition at timestep τ ($e_{i,j} \in E_{\tau+1}$), the set $\mathcal{C}_{\tau+1}$ is equal to $\mathcal{C}_\tau \cup \Phi_{i,j}$, where $\Phi_{i,j} = \{e_{x,y} \notin E_{\tau+1} \mid (v_x, v_y) \in \mathbf{De}(v_j) \times \mathbf{An}(v_i)\}$.*

Proof. The goals are to prove that (1) each candidate edge $e_{x,y} \in \mathcal{C}_{\tau+1}$ would cause a cycle at time $\tau + 1$ and (2) that no other cycle-inducing candidate edge exists. In other words, establishing (1) proves necessity and (2) proves sufficiency.

(1) Necessity. There are two possible cases, corresponding to the components of the union.

Case 1: $e_{x,y} \in \mathcal{C}_\tau$. By definition, the introduction of $e_{x,y}$ causes a cycle at time τ. The fact that the sets of edges in successive timesteps are strict subsets of each other, i.e. $E_\tau \subset E_{\tau+1}$, implies that $e_{x,y}$ would cause a cycle at time $\tau + 1$.

Case 2: $e_{x,y} \notin \mathcal{C}_\tau$. The candidate must belong to the second component of the union, which implies that $v_x \in \mathbf{De}(v_j)$ and $v_y \in \mathbf{An}(v_i)$. There are four subcases, arising from whether $x \overset{?}{=} j$ and $y \overset{?}{=} i$.

Let us treat the subcase ($x \neq j, y \neq i$), noting that the others are provable in a similar manner. By definitions of the descendant and ancestor sets, we have that there exist the paths $(v_j, \dots, v_x)$ and $(v_y, \dots, v_i)$. Given that edge $e_{i,j}$ already exists (having been added previously), the addition of edge $e_{x,y}$ would create the path $(v_i, v_j, \dots, v_x, v_y, \dots, v_i)$, which is a cycle. Hence, $e_{x,y}$ would cause a cycle at time $\tau + 1$.

(2) Sufficiency. Next, the goal is to prove that no other cycle-inducing candidate edge exists. We do this by contradiction: assume that there is an edge $e_{x,y} \notin \mathcal{C}_{\tau+1}$ such that its addition causes a cycle.

By the definition of $\mathcal{C}_{\tau+1}$ and De Morgan's laws for sets, we have that $e_{x,y} \notin \mathcal{C}_t \wedge e_{x,y} \notin \Phi_{i,j}$. The first clause implies that the candidate edge was not cycle-inducing before the addition of $e_{i,j}$ in the last step; therefore, any such cycle must contain the edge $e_{i,j}$ in addition to $e_{x,y}$. The second clause implies that $v_x \notin \mathbf{De}(v_j)$ and $v_y \notin \mathbf{An}(v_i)$, which means that there are no paths of the form $(v_j, \dots, v_x)$ and $(v_y, \dots, v_i)$. Let us use $v_j \nrightarrow v_x$ and $v_y \nrightarrow v_i$ to denote these unreachability constraints. Given the requirement for the cycle to contain both $e_{i,j}$ and $e_{x,y}$, the possible types of cycles are:

1. $(v_i, v_j \nrightarrow v_x, v_y \nrightarrow v_i)$,
2. $(v_x, v_y \nrightarrow v_i, v_j \nrightarrow v_x)$,
3. $(v_{i=x}, v_y \nrightarrow v_{i=x})$,
4. $(v_{x=i}, v_j \nrightarrow v_{x=i})$,
5. $(v_i, v_{j=y} \nrightarrow v_i)$,
6. $(v_x, v_{y=j} \nrightarrow v_x)$.

This list is exhaustive because, by definition, $v_x \notin \mathbf{De}(v_j) \implies x \neq j$ and $v_y \notin \mathbf{An}(v_i) \implies y \neq i$. These paths cannot be cycles as a consequence of the unreachability constraints defined above. Therefore, the edge $e_{x,y} \notin \mathcal{C}_{\tau+1}$ does not cause a cycle, which contradicts our initial assumption. ■

Appendix B. Further experiment details

Code and data: Refer to the repository at https://github.com/VictorDarvariu/causal-discovery-mbrl or the README.pdf file contained in the supplementary material for detailed instructions on how to set up and run the implementation. Given the use of the Docker container software, which fully specifies how dependencies should be obtained and installed, performing the set-up is greatly simplified. For both *Sachs* and *SynTReN*, we use the files released by [21] at https://github.com/kurowasan/GraN-DAG/tree/master/data under the MIT licence. We leverage the synthetic data generator implementations used in [21] for GP regression (MIT licence) and [9] (Apache license) for quadratic regression, respectively. For discrete RV data, we use the

Table 5. Approximate wall clock running time in hours, minutes and seconds with the $\mathcal{F}_{\text{EV}}$ score function.

	CD-UCT	RL-BIC	GS	RS	US	CAM	LiNGAM	NOTEARS	GES	PC
Sachs	1:49:39	1:30:12	0:01:20	2:35:02	0:00:04	0:00:39	0:00:04	0:00:05	0:00:01	0:00:03
SynTReN	2:01:43	35:50:42	0:03:07	9:33:47	0:00:07	0:01:38	0:00:22	0:01:58	0:00:01	0:00:04

Bayesian Network Repository [67]. Datasets are provided in the datasets/subdirectory of the repository and supplementary material to render reproducibility easier.

Implementation details: We build our incremental DAG construction environment and the agents including CD-UCT in pure Python. The current implementation is a prototype and substantial improvements in speed (easily 2–3 orders of magnitude) can be obtained through implementation in a lower-level programming language, especially as graph size grows. We leverage the score function, metrics and pruning implementations released by [9] at https://github.com/huawei-noah/trustworthyAI/tree/master/research/Causal%20Discovery%20with%20RL under the Apache license. For LiNGAM and NOTEARS, we use the open-source implementations provided by their respective authors. For CAM, GES and PC, we use the 'bridge' to implementations in the R programming language from the Causal Discovery Toolbox library [68].

Parameters: For CD-UCT, we vary the exploration parameter $c_{\text{UCT}} \in \{0.025, 0.05, 0.075, 0.1\}$, while for RL-BIC we vary the model input dimension $\in \{64, 128\}$ and learning rate $\in \{0.001, 0.0001\}$—this set includes the default parameters reported in the paper. For the other methods, the default hyperparameters are used. For CD-UCT, we use the full search horizon on *Sachs* and the $d = 10$ synthetic continuous RV graphs and a reduced horizon of $h = 16$ for *SynTReN*, $h = 8$ for the $d = 50$ synthetic graph, $h = 4$ for the discrete RV graphs, respectively. These values were selected by grid search for reward maximization as described below. As the breadth and the depth of search tree grow, a limited horizon is required due to greatly increased search spaces, which lead to substantially noisier estimates of future rewards using uniform random sampling. For the synthetic graph experiments with varying number of datapoints n, the full set of considered values is $n \in \{10, 25, 50, 75, 100, 175, 250, 375, 500, 750, 1000, 1750, 2500, 3750, 5000\}$.

Hyperparameter selection methodology: We perform a simple grid search to determine the best values of the hyperparameters. We note that the goal of hyperparameter selection is to optimize the reward (i.e. find the best graph w.r.t. $\mathcal{F}$) and not the final evaluation metric (e.g. SHD) as this would mean validating on the ground truth graph, which is methodologically incorrect. Furthermore, the initializations of the algorithms differ between the parameter tuning and evaluation runs, so that we do not simply memorize the best results.

Computational infrastructure: Experiments were conducted exclusively on CPUs on an in-house high-performance computing (HPC) cluster. On this infrastructure, the experiments for *Sachs* took approximately 43 days of single-core CPU time, while the experiments for *SynTReN* took approximately 1660 days (approx. 4.5 years) of single-core CPU time. These figures include hyperparameter tuning. The majority of the computational time for *SynTReN* was used by RL-BIC due to its inefficiency (see the 'runtime analysis' paragraph in §5). The experiments for the graphs with discrete RVs took approximately 33 days of single-core CPU time.

Appendix C. Results with $\mathcal{F}_{\text{EV}}$ score function

Results with the $\mathcal{F}_{\text{EV}}$ score function are shown in table 6. Corresponding runtimes are shown in table 5. CD-UCT is still the best method in terms of rewards for both datasets. Interestingly, in the case of the *Sachs* dataset, the best score does not necessarily lead to the best values for the metrics (TPR/FDR/SHD), with CD-UCT, RL-BIC and Random Search all performing similarly in this regard.

Table 6. Results obtained by the methods in the construction phase (top) and after undergoing pruning (bottom) with the $\mathcal{F}_{EV}$ score function.

phase			CD-UCT	RL-BIC	GS	RS	US	CAM	LiNGAM	NOTEARS	GES*	PC*
construct	*Sachs*	reward ↑	$-0.711_{\pm 0.002}$	$-0.726_{\pm 0.035}$	−0.756	$-0.748_{\pm 0.003}$	$-1.409_{\pm 0.018}$	—	—	—	—	—
		TPR ↑	$0.524_{\pm 0.015}$	$0.565_{\pm 0.015}$	0.471	$0.558_{\pm 0.018}$	$0.421_{\pm 0.030}$	—	—	—	—	—
		FDR ↓	$0.818_{\pm 0.005}$	$0.814_{\pm 0.004}$	0.837	$0.806_{\pm 0.006}$	$0.854_{\pm 0.010}$	—	—	—	—	—
		SHD ↓	$42.070_{\pm 0.438}$	$43.250_{\pm 0.689}$	41.000	$41.220_{\pm 0.438}$	$43.700_{\pm 0.587}$	—	—	—	—	—
	SynTReN	reward ↑	$-0.296_{\pm 0.011}$	$-0.835_{\pm 0.101}$	−0.361	$-0.506_{\pm 0.017}$	$-1.302_{\pm 0.038}$	—	—	—	—	—
		TPR ↑	$0.438_{\pm 0.021}$	$0.318_{\pm 0.032}$	0.332	$0.317_{\pm 0.015}$	$0.256_{\pm 0.014}$	—	—	—	—	—
		FDR ↓	$0.894_{\pm 0.005}$	$0.926_{\pm 0.006}$	0.917	$0.923_{\pm 0.004}$	$0.938_{\pm 0.004}$	—	—	—	—	—
		SHD ↓	$93.565_{\pm 0.938}$	$102.075_{\pm 5.550}$	96.300	$99.805_{\pm 0.833}$	$102.155_{\pm 0.899}$	—	—	—	—	—
prune	*Sachs*	TPR ↑	$0.334_{\pm 0.013}$	$0.334_{\pm 0.011}$	0.235	$0.339_{\pm 0.016}$	$0.211_{\pm 0.021}$	0.353	0.235	0.118	0.353	0.353
		FDR ↓	$0.388_{\pm 0.022}$	$0.331_{\pm 0.018}$	0.600	$0.369_{\pm 0.027}$	$0.577_{\pm 0.039}$	0.400	0.556	0.500	0.250	0.250
		SHD ↓	$12.150_{\pm 0.217}$	$11.730_{\pm 0.199}$	14.000	$12.040_{\pm 0.257}$	$14.060_{\pm 0.352}$	12.000	14.000	15.000	11.000	11.000
	SynTReN	TPR ↑	$0.297_{\pm 0.017}$	$0.168_{\pm 0.015}$	0.234	$0.234_{\pm 0.014}$	$0.189_{\pm 0.012}$	0.329	0.303	×	0.460	0.337
		FDR ↓	$0.822_{\pm 0.010}$	$0.873_{\pm 0.011}$	0.854	$0.856_{\pm 0.009}$	$0.879_{\pm 0.008}$	0.794	0.809	×	0.832	0.749
		SHD ↓	$44.020_{\pm 1.119}$	$43.500_{\pm 1.167}$	45.800	$46.890_{\pm 1.120}$	$47.715_{\pm 0.946}$	41.300	41.600	×	61.700	37.800

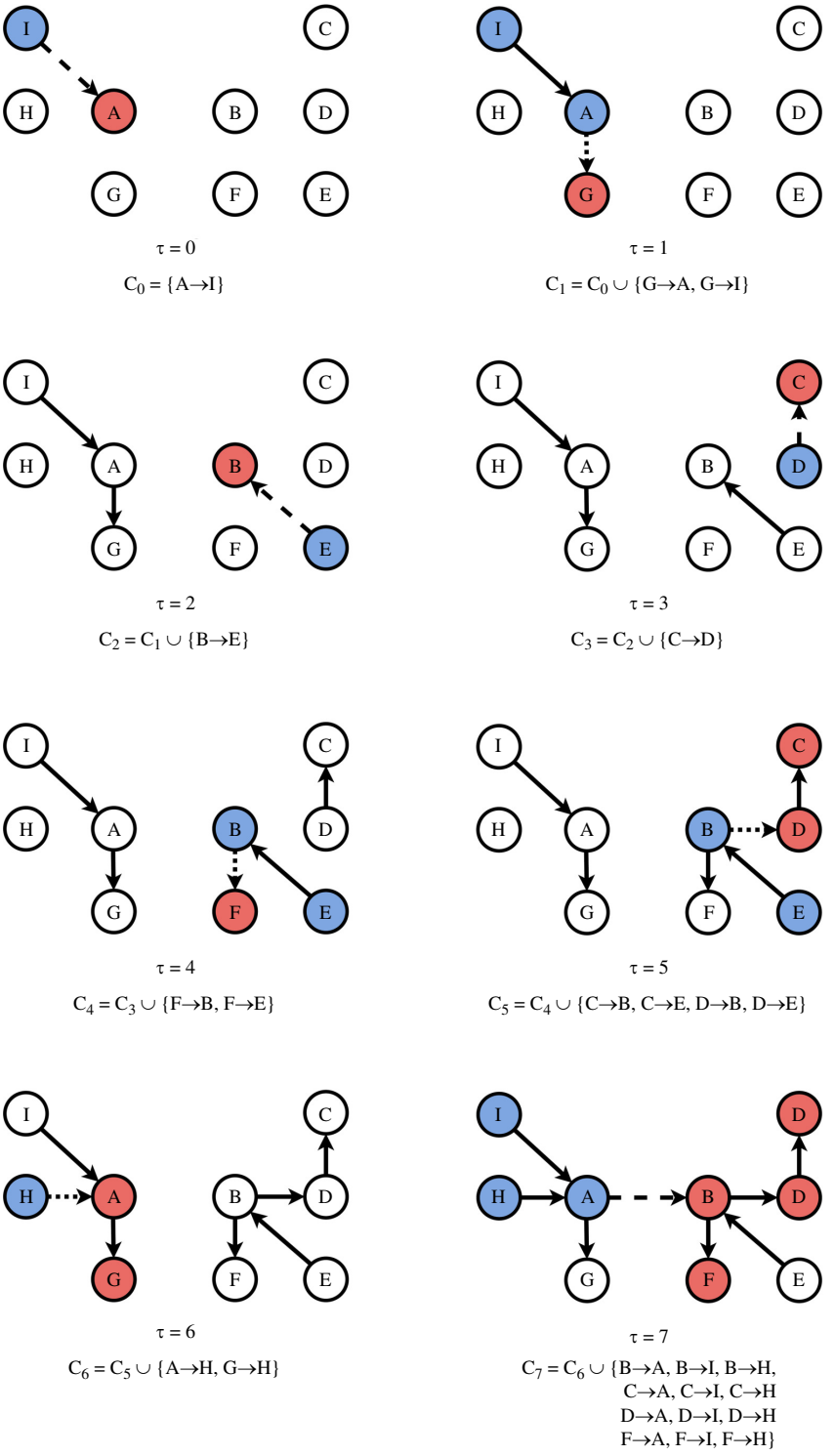

Figure 5. A full illustration of the proposed algorithm for tracking cycle-inducing edges. At each timestep τ, an edge (shown with a dashed line) is introduced, and the candidate edges that would connect a *descendant* of the endpoint to an *ancestor* of the starting point are added to the set $\mathcal{C}_\tau$. This eliminates the need to explicitly check for cycles. We also note that this algorithm simply determines the *invalid* edges, with the choice of which edge to add being left to the higher-level causal discovery method.

References

1. Friedman N, Linial M, Nachman I, Pe'er D. 2000 Using Bayesian networks to analyze expression data. In *Int. Conf. on Computational Molecular Biology*. (doi:10.1145/332306.332355)
2. Morgan SL, Winship C. 2015 *Counterfactuals and causal inference*. Cambridge, UK: Cambridge University Press.
3. Pearl J. 2009 *Causality*. Cambridge, UK: Cambridge University Press.
4. Peters J, Janzing D, Schölkopf B. 2017 *Elements of causal inference: foundations and learning algorithms*. Cambridge, MA: MIT Press.
5. Schwarz G. 1978 Estimating the dimension of a model. *Ann. Stat.* **6**, 461–464.
6. Chickering M, Heckerman D, Meek C. 2004 Large-sample learning of Bayesian networks is NP-hard. *J. Mach. Learn. Res.* **5**, 1287–1330.
7. Shimizu S, Hoyer PO, Hyvärinen A, Kerminen A, Jordan M. 2006 A linear non-Gaussian acyclic model for causal discovery. *J. Mach. Learn. Res.* **7**, 2003–2030.
8. Huang B, Zhang K, Lin Y, Schölkopf B, Glymour C. 2018 Generalized score functions for causal discovery. In *Proc. 24th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August*. New York, NY: Association for Computing Machinery
9. Zhu S, Ng I, Chen Z. 2020 Causal discovery with reinforcement learning. In *Proc. 8th Int. Conf. on Learning Representations (ICLR 2020), Virt. Conf., 26 April–1 May*. OpenReview
10. Darvariu VA, Hailes S, Musolesi M. 2023 Planning spatial networks with Monte Carlo tree search. *Proc. R. Soc. A.* **479**, 20220383. (doi:10.1098/rspa.2022.0383)
11. Kocsis L, Szepesvári C. 2006 Bandit based Monte-Carlo planning. In *Proc. 17th Euro. Conf. on Machine Learning (ECML 2006), Berlin, Germany, 18–22 September*. Berlin/Heidelberg, Germany: Springer-Verlag. (doi:10.1007/11871842_29)
12. Chickering DM. 2002 Optimal structure identification with greedy search. *J. Mach. Learn. Res.* **3**, 507–554.
13. Koivisto M, Sood K. 2004 Exact Bayesian structure discovery in Bayesian networks. *J. Mach. Learn. Res.* **5**, 549–573. (doi:10.48550/arXiv.1206.6828)
14. Jaakkola T, Sontag D, Globerson A, Meila M. 2010 Learning Bayesian network structure using LP relaxations. In *Proc. 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010), Sardinia, Italy, 13–15 May*. PMLR.
15. Scanagatta M, de Campos CP, Corani G, Zaffalon M. 2015 Learning Bayesian networks with thousands of variables. In *Proc. 29th Conf. on Neural Information Processing Systems (NeurIPS 2015), Vancouver, Canada, 7–12 December*. Red Hook, NY: Curran Associates.
16. Ramsey J, Glymour M, Sanchez-Romero R, 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. *Int. J. Data Sci. Analyt.* **3**, 121–129. (doi:10.1007/s41060-016-0032-z)
17. Chickering DM, Geiger D, Heckerman D. 1995 Learning Bayesian networks: search methods and experimental results. In *Pre-proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics* Proceedings of Machine Learning Research, pp. 112–128. PMLR.
18. Meek C. 1997 *Graphical Models: Selecting causal and statistical models*. PhD thesis. Carnegie Mellon University.
19. Zheng X, Aragam B, Ravikumar PK, Xing EP. 2018 DAGs with no tears: continuous optimization for structure learning. In *Proc. 32nd Conf. on Neural Information Processing Systems (NeurIPS 2018), Montreal, Canada, 2–8 December*. Red Hook, NY: Curran Associates.
20. Yu Y, Chen J, Gao T, Yu M. 2019 DAG-GNN: DAG structure learning with graph neural networks. In *Proc. 36th Int. Conf. on Machine Learning (ICML 2019), Long Beach, CA, 9–15 June*. PMLR.
21. Lachapelle S, Brouillard P, Deleu T, Lacoste-Julien S. 2020 Gradient-based neural DAG learning. In *Proc. 8th Int. Conf. on Learning Representations (ICLR 2020), Virt. Conf., 26 April–1 May*. OpenReview.
22. Ng I, Huang B, Zhang K. 2024 Structure learning with continuous optimization: A sober look and beyond. In *Proc. 3rd Conf. on Causal Learning and Reasoning (CLeaR 2024), Los Angeles, CA, 1–3 April*. PMLR.
23. Friedman N, Koller D. 2003 Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Mach. Learn.* **50**, 95–125.

24. Bühlmann P, Peters J, Ernest J. 2014 CAM: Causal additive models, high-dimensional order search and penalized regression. *Ann. Stat.* **42**, 2526–2556. (doi:10.1214/14-aos1260)
25. Charpentier B, Kibler S, Günnemann S. 2022 Differentiable DAG sampling. In *Proc. 10th Int. Conf. on Learning Representations (ICLR 2022), Virt. Conf., 25–29 April*. OpenReview.
26. Zantedeschi V, Franceschi L, Kaddour J, Kusner MJ, Niculae V. 2023 DAG learning on the permutahedron. In *Proc. 11th Int. Conf. on Learning Representations (ICLR 2023), Kigali, Rwanda, 1–5 May*. OpenReview.
27. Wang X, Du Y, Zhu S, Ke L, Chen Z, Hao J, Wang J. 2021 Ordering-based causal discovery with reinforcement learning. In *Proc. 30th Int. Joint Conf. on Artificial Intelligence (IJCAI-21), Virt. Conf., 19–26 August*. International Joint Conferences on Artificial Intelligence. (doi:10.24963/ijcai.2021/491)
28. Yang D, Yu G, Wang J, Wu Z, Guo M. 2023 Reinforcement causal structure learning on order graph. In *Proc. 37th AAAI Conf. on Artificial Intelligence (AAAI-23), Washington, DC, 7–14 February*. Palo Alto, CA: AAAI Press.
29. Bello I, Pham H, Le QV, Norouzi M, Bengio S. 2017 Neural combinatorial optimization with reinforcement learning. In *Proc. 5th Int. Conf. on Learning Representations (ICLR 2017) Workshop Track, Toulon, France, 24–26 April*. OpenReview.
30. Khalil EB, Dai H, Zhang Y, Dilkina B, Song L. 2017 Learning combinatorial optimization algorithms over graphs. In *Proc. 31st Conf. on Neural Information Processing Systems (NeurIPS 2017), Long Beach, CA, 4–9 December*. Red Hook, NY: Curran Associates.
31. Dai H, Li H, Tian T, Huang X, Wang L, Zhu J, Song L. 2018 Adversarial attack on graph structured data. In *Proc. 35th Int. Conf. on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July*. PMLR.
32. You J, Liu B, Ying R, Pande V, Leskovec J. 2018 Graph convolutional policy network for goal-directed molecular graph generation. In *Proc. 32nd Conf. on Neural Information Processing Systems (NeurIPS 2018), Montreal, Canada, 2–8 December*. Red Hook, NY: Curran Associates.
33. Li Y, Vinyals O, Dyer C, Pascanu R, Battaglia P. 2018 Learning deep generative models of graphs. In *Proc. 35th Int. Conf. on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July*. PMLR.
34. Darvariu VA, Hailes S, Musolesi M. 2021 Goal-directed graph construction using reinforcement learning. *Proc. R. Soc. A.* **477**, 20210168. (doi:10.1098/rspa.2021.0168)
35. Russell SJ, Norvig P. 2020 *Artificial intelligence: a modern approach*, 4th edn. Upper Saddle River, NJ: Prentice Hall.
36. Cormen TH, Leiserson CE, Rivest RL, Stein C. 2022 *Introduction to algorithms*, 4th edn. Cambridge, MA: MIT Press.
37. Berliner HJ. 1973 Some necessary conditions for a master chess program. In *Proc. 3rd Int. Joint Conf. on Artificial Intelligence (IJCAI-73), Stanford, CA, 20–23 August*. International Joint Conferences on Artificial Intelligence.
38. Abramson B. 1990 Expected-outcome: a general model of static evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.* textbf12, 182–193. (doi:10.1109/34.44404)
39. Tesauro G, Galperin GR. 1997 On-line policy improvement using Monte-Carlo search. In *NeurIPS'97*.
40. Sutton RS, Barto AG. 2018 *Reinforcement learning: an introduction*. Cambridge, MA: MIT Press.
41. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S. 2012 A survey of Monte Carlo Tree search methods. *IEEE Trans. Comput. Intell. AI Games* **4**, 1–43. (doi:10.1109/tciaig.2012.2186810)
42. Auer P, Cesa-Bianchi N, Fischer P. 2002 Finite-time analysis of the multiarmed Bandit problem. *Mach. Learn.* **47**, 235–256.
43. Guo X, Singh S, Lee H, Lewis RL, Wang X. 2014 Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Proc. 28th Conf. on Neural Information Processing Systems (NeurIPS 2014), Montreal, Canada, 8–13 December*. Red Hook, NY: Curran Associates.
44. Anthony T, Tian Z, Barber D. 2017 Thinking fast and slow with deep learning and tree search. In *NeurIPS'17*.
45. Hoyer P, Janzing D, Mooij JM, Peters J, Schölkopf B. 2008 Nonlinear causal discovery with additive noise models. In *Proc. 22nd Conf. on Neural Information Processing Systems (NeurIPS 2008), Montreal, Canada, 8–13 December*. Red Hook, NY: Curran Associates.
46. Koller D, Friedman N. 2009 *Probabilistic graphical models: principles and techniques*. Cambridge, MA: MIT Press.

47. De Campos LM, Friedman N. 2006 A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *J. Mach. Learn. Res.* **7**.
48. Rasmussen CE, Williams CK. 2006 *Gaussian processes for machine learning*. Cambridge, MA: MIT Press.
49. Aho AV, Hopcroft JE, Ullman JD. 1974 *The design and analysis of computer algorithms*. Boston, MA: Addison-Wesley.
50. Purdom Jr. P. 1970 A transitive closure algorithm. *BIT Numer. Math.* **10**, 76–94. (doi:10.1007/bf01940892)
51. Italiano GF. 1986 Amortized efficiency of a path retrieval data structure. *Theoret. Comput. Sci.* **48**, 273–281. (doi:10.1016/0304-3975(86)90098-8)
52. La Poutré JA, van Leeuwen J. 1989 Maintenance of transitive closures and transitive reductions of graphs. Technical Report RUU-CS-87-25, University of Utrecht. (doi:10.1007/3-540-19422-3_9)
53. Sachs K, Perez O, Pe'er D, Lauffenburger DA, Nolan GP. 2005 Causal protein-signaling networks derived from multiparameter single-cell data. *Science* **308**, 523–529. (doi:10.1126/science.1105809)
54. van den Bulcke T, Van Leemput K, Naudts B, van Remortel P, Ma H, Verschoren A, De Moor B, Marchal K. 2006 SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics* **7**, 1–12. (doi:10.1186/1471-2105-7-43)
55. Reisach A, Seiler C, Weichwald S. 2021 Beware of the simulated DAG! Causal discovery benchmarks may be easy to game. In *Proc. 35th Conf. on Neural Information Processing Systems (NeurIPS 2021), Virt. Conf., 6–14 December*. Red Hook, NY: Curran Associates. (doi:10.1111/j.2517-6161.1988.tb01721.x)
56. Lauritzen SL, Spiegelhalter DJ. 1988 Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc. B Methodol.* **50**, 157–194.
57. Spiegelhalter DJ. 1992 Learning in probabilistic expert systems. *Bayesian Stat.* **4**, 447–465.
58. Binder J, Koller D, Russell S, Kanazawa K. 1997 Adaptive probabilistic networks with hidden variables. *Mach. Learn.* **29**, 213–244.
59. Bühlmann P, Van De Geer S. 2011 *Statistics for high-dimensional data: methods, theory and applications*. Berlin, Heidelberg, Germany: Springer. (doi:10.1007/978-3-642-20192-9)
60. Cussens J. 2011 Bayesian network learning with cutting planes. In *Proc. 27th Conf. on Uncertainty in Artificial Intelligence (UAI 2011), Barcelona, Spain, 14–17 July*. Corvallis, OR: AUAI Press.
61. Verma TS, Pearl J. 1991 Equivalence and synthesis of causal models. In *Proc. 7th Conf. on Uncertainty in Artificial Intelligence (UAI 1991), Los Angeles, CA, 13–15 July*. San Mateo, CA: Morgan Kaufmann.
62. Spirtes P, Glymour C, Scheines R. 2000 *Causation, prediction, and search*. Cambridge, MA: MIT Press. (doi:10.7551/mitpress/1754.001.0001)
63. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap TP, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. 2016 Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489. (doi:10.1038/nature16961)
64. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M. 2019 Learning scheduling algorithms for data processing clusters. In *Proc. 2019 Conf. of the ACM Special Interest Group on Data Communication (SIGCOMM), Beijing, China, 19–24 August*. New York, NY: Association for Computing Machinery.
65. Martin RC. 2000 Design principles and design patterns. *Object Mentor* **1**, 597.
66. Borenstein D. 2000 A directed acyclic graph representation of routing manufacturing flexibility. *Eur. J. Oper. Res.* **127**, 78–93. (doi:10.1016/s0377-2217(99)00324-0)
67. Elidan G. 2001 Bayesian network repository. See https://www.cs.huji.ac.il/~galel/Repository/. (accessed 12 February 2024).
68. Kalainathan D, Goudet O, Dutta R. 2020 Causal discovery toolbox: uncovering causal relationships in Python. *J. Mach. Learn. Res.* **21**, 1406–1410.
69. Darvariu VA, Hailes S, Musolesi M. 2025 Supplementary material from: Tree Search in DAG Space with Model-based Reinforcement Learning for Causal Discovery. Figshare (doi:10.6084/m9.figshare.c.7665971)