Proceedings of Machine Learning Research 275:1–27, 2025 4th Conference on Causal Learning and Reasoning

Exact discovery is polynomial for certain sparse causal Bayesian networks

Felix L. Rios and **Giusi Moffa**
Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland

Jack Kuipers JACK.KUIPERS@BSSE.ETHZ.CH
D-BSSE, ETH Zurich, Basel, Switzerland

Editors: Biwei Huang and Mathias Drton

Abstract

Causal Bayesian networks are widely used tools for summarising the dependencies between variables and elucidating their putative causal relationships. By restricting the search to trees, for example, learning the optimum from data is polynomial, but this does not guarantee finding the optimal network overall. Without similar restrictions, exact discovery of the optimum is computationally hard in general and no polynomial results are known. The current state-of-the-art approaches are integer linear programming over the underlying space of directed acyclic graphs, dynamic programming and shortest-path searches over the space of topological orders, and constraint programming combining both. For dynamic programming over orders, the computational complexity is known to be exponential base 2 in the number of variables in the network. We demonstrate how to use properties of Bayesian networks to prune the search space and lower the computational cost, while still guaranteeing exact discovery of the provably optimal network. We also include new path-search and divide-and-conquer criteria. Without *a priori* constraining the search to certain types of networks, the algorithm completes in quadratic time when the optimum is a matching, and in polynomial time when the optimum belongs to any network class with logarithmically-bound largest connected components. In simulation studies we observe the polynomial dependence for sparse networks and that, beyond some critical value, the logarithm of the base grows with the network density. Our approach then out-competes the state-of-the-art at lower densities. These results therefore pave the way for faster exact causal discovery in larger and sparser networks.

Keywords: Causal Bayesian networks, Structure learning, Exact algorithms, Complexity

1. Introduction

Causal Bayesian networks have in the last decades become standard models for describing causal relationships among a set of random variables since their directed edges may provide a powerful handle for complex causal queries. In line with increased general interest in causality, the interest in automatic procedures for learning the structure of Bayesian networks encoding the probabilistic dependencies of causal mechanisms, so-called *causal structure learning* or simply *causal discovery*, has also increased. Although finding the best tree is polynomial (Chow and Liu, 1968), this does not prove that the tree is optimal amongst all networks and hence does not achieve exact discovery of the optimal network. Searching within similarly restricted spaces like networks which can be turned into trees or branchings with a fixed number of edge of vertex deletions (Gaspers et al., 2015; Grüttemeier and Komusiewicz, 2022) is also polynomial. However, exact discovery without such restrictions is in general NP-hard (Chickering, 1996; Chickering et al., 2004) and no polynomial results are known. For this reason, most existing algorithms are based on heuristic or approximate

© 2025 F.L. Rios, G. Moffa & J. Kuipers.

methods, and a plethora of different approaches and implementations exist (Scutari et al., 2019; Rios et al., 2021). These are usually either *score-based*, where each network has a goodness of fit score to the data, *constraint-based*, where networks are learned by independence testing, or *hybrid* combining both aspects. With the recent development of software packages (Rios et al., 2021) for using and comparing algorithms, we can expect the pace of advances to continue to increase.

In this paper, we wish to examine the NP-hardness in detail and focus on exact score-based causal discovery. The current state-of-the-art approaches (Malone et al., 2018) are integer linear programming (ILP) over the underlying space of directed acyclic graphs (DAGs) (Bartlett and Cussens, 2017; Cussens et al., 2017; Cussens, 2020), dynamic programming (Koivisto and Sood, 2004; Silander et al., 2006) and path searches (Yuan and Malone, 2013) over the space of topological orders (Friedman and Koller, 2003; Teyssier and Koller, 2005). Finally, constraint programming approaches use both spaces (van Beek and Hoffmann, 2015; Trösser et al., 2021), and compared to dynamic programming, they prune networks that can never be optimal.

Here, we present an algorithm for exact causal discovery with new pruning rules for sequentially eliminating suborders. Also included are alternative bounds on path searches (Yuan and Malone, 2013), computable at low complexity, further allowing us to prove optimality in polynomial time for matchings (graphs with at most one neighbour for each node). Combining our pruning rules with a new divide-and-conquer algorithm, we can extend our polynomial exact discovery to classes of sparse graphs. The rest of the paper is structured as follows. Section 2 reviews order-based exact structure learning. Section 3 describes our new pruning approach which we use to prove optimality for matchings and certain classes of sparse graphs in polynomial time in Section 4. In Section 5 we empirically demonstrate the polynomial dependence of our method, and how the hardness depends on the network density beyond some critical point. Section 6 provides a discussion of our results.

1.1. Preliminaries

Let $\mathcal{G} = (V, E)$ be a DAG with vertex set $V = \{1, \ldots, p\}$ and edge set E. The vertices correspond to the random variables in a Bayesian network and the edges to the direct dependence between pairs of nodes. Following the factorisation of the joint probability distribution of a Bayesian network as $P(V) = \prod_{i=1}^{p} P(i \mid \mathbf{Pa}_i)$, where $\mathbf{Pa}_i$ are the parents of node i, the log-score of a network is

$$\sigma(\mathcal{G}) = \sum_{i=1}^{p} \sigma(i, \mathbf{Pa}_i \mid D) \tag{1}$$

where σ is a function of the data D depending only on a node and its parents. Examples include penalised likelihoods, like the BIC, or marginalised likelihoods, like the BDe for categorical data (Heckerman and Geiger, 1995) and the BGe for linear-Gaussian data (Geiger and Heckerman, 2002; Kuipers et al., 2014).

The task of exact causal discovery is to find a DAG that maximises the score: $\arg\max_{\mathcal{G}} \sigma(\mathcal{G})$. Each DAG belongs to at least one topological *order*, $v_{1:p} = \langle v_1, \ldots, v_p \rangle$, of the nodes where the edges are pointing to the left. The score of a node v_j in an order depends on a subset of the nodes further right, referred to as *potential parents*. We denote the maximum possible node score by

$$s(v_j | v_{j+1:p}) = \max_{\mathbf{Pa}_{v_j} \subseteq \{v_{j+1:p}\}} \sigma(v_j, \mathbf{Pa}_{v_j} \mid D) \tag{2}$$

As a consequence, more potential parents for a specific node cannot decrease its score. We consider the setting where finding the optimal parent set for each node in the network can be performed in

polynomial time, for example by limiting the in-degree, so that the complexity bottleneck is in the network search which is generally NP-hard (Chickering, 1996; Chickering et al., 2004). We assign to each order the score of an optimal DAG consistent with that order

$$S(v_{1:p}) := \sum_{j=1}^{p} s(v_j | v_{j+1:p}). \tag{3}$$

Finding an order with an optimal score then provides a maximally scoring DAG (Koivisto and Sood, 2004; Teyssier and Koller, 2005). Note that different orders of the same set of nodes might give the same score, implying that there might be more than one optimal ordering. Since we only need to find one, we distinguish an optimal order (OO) from an ordered optimal order (OOO) where any neighbouring nodes in an OO with the same score on transposition are in numerical order.

Our task is to find the OOO, and hence a DAG, with the maximum score. We focus on finding the highest-scoring network in a restricted search space where each node has a preselected set of parents, up to total size K and one additional parent outside that set. One way to obtain the search space is with a hybrid approach, starting from a constraint-based skeleton and iteratively improving it with search-and-score (Kuipers et al., 2022). We focus on this setting since accessing the relevant scores is highly efficient, while the iterative approach of updating the search space, and subsequent sampling of DAGs (Kuipers et al., 2022), has been shown to offer good performance in causal discovery benchmarks (Rios et al., 2021) as well as enabling Bayesian analyses.

We distinguish between two types of *suborders* of length n, *left* and *right orders*, denoted as $\overrightarrow{v}_{1:n} = \langle v_{1:n}, [\dots] \rangle$ and $\overleftarrow{v}_{n:1} = \langle [\dots], v_{n:1} \rangle$, respectively. For tidiness, we remove brackets in the subscripts so $v_{(a+b):(c+d)} =: v_{a+b:c+d}$. The square bracket notation symbolises the set of the nodes outside the order, $V \setminus \{v_j\}_{j=1}^{n}$, which we refer to as *dormant*. Each suborder represents the set of orders matching the specific ordering of the *visible* (*i.e.* not dormant) nodes, thus for an order v^* we may write statements like $v^* \in \langle [\dots], v_{n:1} \rangle$. The dormant nodes have no internal order. However, when needed we may explicitly label them in the notation as $\langle [h], v_{n:1} \rangle$. Finally, we call the left- and right-most nodes in a suborder the *front* and *back*, respectively.

Definition 1 *The score of a (sub)order is the sum of the scores for the individual visible nodes.*

For left orders, say, we truncate the sum in Equation (3) to $j \leq n$. We let $s_{\max}(\{A\}|B)$ denote the maximal score among the orderings of a set of nodes A where B are potential parents of all nodes in A.

2. Exact causal discovery

Here we present a version of the exact causal discovery algorithm of Koivisto and Sood (2004), rephrased in terms of right (left) orders that we will need later.

2.1. Permutation trees

Since the score of each node only depends on potential parents, we can compute the score of each right (left) order by adding new nodes to the front (back) one at time. This creates a tree of all $p!$ orders, from which we can exhaustively find an optimal order. To reduce complexity, one can use the fact that optimality of an order is inherited by its suborders:

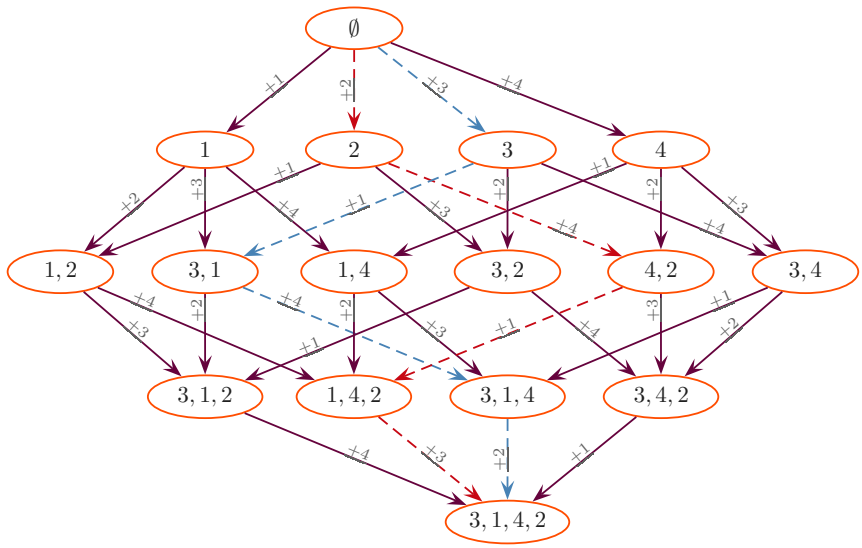


Figure 1: **The ordered power set representation of pruning all $p!$ orders to 2^p ordered subsets.** Moving down the diagram we append the new node indicated on the edge to the front (back) of the previous right (left) suborders. We compare the incoming suborders at each (orange) node and only keep the best. Arriving at the bottom of the diagram we have the OOO. The dashed paths arrive at the OOO when appending at the front (red) or back (blue).

Theorem 2 *For a given optimal right (left) order $\overleftarrow{v}_{n:1}$ ($\overrightarrow{v}_{1:n}$) of $U \subset V$, the right (left) suborder $\overleftarrow{v}_{n-1:1}$ ($\overrightarrow{v}_{1:n-1}$) is an optimal right (left) suborder of $U \setminus \{v_n\}$.*

Proof Suppose $\overleftarrow{v}_{n-1:1}$ is not an optimal right suborder of $U \setminus \{v_n\}$. Then there exists a suborder $\overleftarrow{w}_{n-1:1}$ with $S(\overleftarrow{w}_{n-1:1}) > S(\overleftarrow{v}_{n-1:1})$. Appending v_n in front

$$S(\langle[\ldots], v_n, w_{n-1:1}\rangle) = s(v_n|w_{n-1:1}) + S(\overleftarrow{w}_{n-1:1}) > s(v_n|w_{n-1:1}) + S(\overleftarrow{v}_{n-1:1}) = S(\overleftarrow{v}_{n:1})$$

then $\overleftarrow{v}_{n:1}$ is not optimal, which is a contradiction. The proof for left suborders is analogous. ■

Corollary 3 *For a given optimal right (left) suborder $\overleftarrow{v}_{n:1}$ ($\overrightarrow{v}_{1:n}$) of $U \subset V$, the right (left) suborder $\overleftarrow{v}_{n-k:1}$ ($\overrightarrow{v}_{1:n-k}$) for $1 \leq k \leq n$ is an optimal right (left) suborder of $U \setminus \{v_j\}_{j=n-k+1}^{n}$.*

Proof Apply Theorem 2 several times. ■

2.2. Ordered power set

To reduce the complexity from factorial, Koivisto and Sood (2004) essentially used Theorem 2 to prune suborders with the same set of nodes by comparing their scores and keeping only one with the highest score. We illustrate this in Figure 1 where we project the permutation tree onto the ordered power set or Hasse diagram of the node subsets.

At stage n, we have the $\binom{p}{n-1}$ orders from stage $(n-1)$ each appended with $(p-n+1)$ new elements to create $(p-n+1)\binom{p}{n-1} = p\binom{p-1}{n-1} = n\binom{p}{n}$ suborders of length n.

Pruning 1 (Suborder pruning) *At stage n, we go through each group of n suborders with the same set of nodes and keep only one with the highest score. If several suborders have the same score, we retain the one with the last two nodes in numerical order (lower node number to the left).*

Theorem 4 *The OOO will not be pruned.*

Proof By Theorem 2, any suborder of the OOO will be amongst the highest scoring on that set of nodes, while the transposable nodes of the OOO are in numerical order by construction. ■

Corollary 5 *The resulting order at the last stage of the tree pruning is the OOO.*

This is, for example, the ordered subset at the bottom of the power set of Figure 1.

2.3. Complexity

In the setting of Koivisto and Sood (2004), since the power set is complete we can compare the scores of all the $n\binom{p}{n}$ suborders at stage n in linear time using their indexing in the Hasse diagram, giving a total complexity of $\mathcal{O}(p2^p)$ for the tree pruning. In addition, the lookup of each score when appending a new node is assumed to be $\mathcal{O}(1)$, so scoring all the $n\binom{p}{n}$ new suborders at stage n also leads to a total complexity of $\mathcal{O}(p2^p)$. The alternative approach of Silander et al. (2006) which exhaustively checks for the best sinks to build up an OO from the front likewise has complexity of $\mathcal{O}(p2^p)$. The problem can also be recast as a reinforcement learning task (Mokhtarian et al., 2023), though this also does not reduce the complexity.

Koivisto and Sood (2004) showed how to use the Möbius transform to reduce the complexity to $\mathcal{O}(2^p)$, however this relies on precomputing various score tables. They allowed each node to have up to K possible parents from any of the p available and computing the score table takes $\mathcal{O}(C(m,p,K)p^{K+1})$, where C is the time of each local score computation, possibly depending on the data size m. Thereafter scores are combined giving an overall complexity of $\mathcal{O}(p2^p)$.

In our setting we allow K parents from a preselected set so that creating the original score tables is $\mathcal{O}(C(m,p,K)p2^K)$ and computing combined score tables is $\mathcal{O}(p2^K)$ (Kuipers et al., 2022). In order to not fully trust the preselected search space, we allow one additional parent outside. Appending a new node to a suborder involves looking up $\mathcal{O}(p)$ scores. After the score tables have been created, running through the power set and scoring the suborders leads to an overall complexity of $\mathcal{O}(p^2 2^p)$, which will be the baseline to compare to when we start pruning.

3. Lowering the complexity of exact causal discovery

In the following we show that we can build the full optimal order starting with the empty order, and that at each step n we can prune the suborders of length n that will not lead to the optimal order. In particular, when we add nodes we only need to keep those whose placement is optimal.

Theorem 6 *Let $M_0 = ([\dots])$ and let for each $n \leq p$, M_n be the set of right (left) orders of length n created by adding a node at the front (back), only if that node is optimal there, of some order in M_{n-1}. Then each M_n contains the suborder of an optimal order $v^* \in M_p$.*

Proof We prove inductively that each M_n contains the suborder of an optimal order $v^* \in M_p$. For $n = 1$, every node is optimal at the back so $M_1 = \{\langle[\dots], j\rangle\}_{j=1}^p$ and it is clear that some element here is a suborder of the optimal order. For $n > 1$ we assume inductively that M_{n-1} contains the suborder of the optimal full order. Let $\overleftarrow{v}_{n:1}$ be a suborder of v^*. Then we need to prove that it can be constructed from some order in M_{n-1}. Clearly, $\overleftarrow{v}_{n:1}$ is created from $\overleftarrow{v}_{n-1:1}$ by adding v_n at the back. Since $\overleftarrow{v}_{n:1}$ is a suborder of v^* we know by Corollary 1 that it is optimal on $\{v_j\}_{j=1}^n$. By Theorem 1, $\overleftarrow{v}_{n-1:1}$ is optimal on $\{v_j\}_{j=1}^{n-1}$ so by the induction hypothesis it must be contained in M_{n-1}. ■

Corollary 7 *A full OO can be built by only adding new nodes to the front (back), starting from the empty order.*

Proof By Theorem 6, M_p will contain the optimal order. ■

3.1. Pruning right orders

We focus on pruning specifically for right suborders. If a node added to the front of a right order were to fit better further along the order, then we can prune by Theorem 6.

Pruning 2 (Optimal front) *We prune $\overleftarrow{v}_{n:1}$ if $S(\langle[\dots], v_{n-1:k}, v_n, v_{k-1:1}\rangle) > S(\overleftarrow{v}_{n:1})$ for some $k < n$.*

We can further ensure numerical ordering in the case of equal score under transposition.

Pruning 3 (Ordered front) *We prune $\overleftarrow{v}_{n:1}$ if $v_n > v_{n-1}$ and $S(\overleftarrow{v}_{n:1}) = S(\langle[\dots], v_{n-1}, v_n, v_{n-2:1}\rangle)$.*

where $>$ is the usual mathematical operator since the vertex set just consists of the numbers $1, \dots, p$. Both these pruning rules have been employed previously (van Beek and Hoffmann, 2015). For further pruning, however, we use the following observation:

Lemma 8 *For any partition $a \cup b$ of the dormant nodes in a right suborder $\overleftarrow{v}_{n:1}$ it holds that*

$$s_{\max}(\{a \cup b\}|v_{n:1}) \leq s_{\max}(\{a\}|b, v_{n:1}) + s_{\max}(\{b\}|a, v_{n:1}). \tag{4}$$

Proof On the right the nodes in a and b are allowed all the nodes in b and a as parents respectively, while on the left there are restrictions induced by the ordering of $a \cup b$, so those scores can only be lower or equal. ■

Theorem 9 *If $S(\overleftarrow{v}_{n:1}) = S(\langle v_n, [\dots], v_{n-1:1}\rangle)$ then $\max_{v \in \langle[\dots], v_{n-1:1}\rangle} S(v) \leq \max_{v \in \langle[\dots], v_{n:1}\rangle} S(v)$.*

Proof We denote by $\boldsymbol{h} = V \setminus \{v_j\}_{j=1}^n$ the dormant nodes of $\overleftarrow{v}_{n:1}$. The maximal score of an order in $\overleftarrow{v}_{n-1:1}$ is

$$\begin{aligned}
&\max_{v \in \langle[\dots], v_{n-1:1}\rangle} S(v) = s_{\max}(\{\boldsymbol{h}, v_n\}|v_{n-1:1}) + S(\overleftarrow{v}_{n-1:1}) \\
&\leq s_{\max}(\{\boldsymbol{h}\}|v_{n:1}) + s(v_n|\boldsymbol{h}, v_{n-1:1}) + S(\overleftarrow{v}_{n-1:1}) \text{ / by (4)} \\
&= s_{\max}(\{\boldsymbol{h}\}|v_{n:1}) + S(\langle v_n, [\dots], v_{n-1:1}\rangle) = s_{\max}(\{\boldsymbol{h}\}|v_{n:1}) + S(\overleftarrow{v}_{n:1}) = \max_{v \in \langle[\dots], v_{n:1}\rangle} S(v).
\end{aligned}$$

■

Theorem 9 means that if the score of v_n is independent of the dormant nodes, the total order score cannot be improved by putting some other node at the front. Since we have a numerical ordering constraint in the case of equal scores, we can prune for any node with a lower number and only keep particles with the highest number node in front whose score is independent of the dormant nodes.

Pruning 4 (No right gaps) *Consider all the right orders of length n created from $\overleftarrow{v}_{n-1:1}$. Let v'_n be the maximal node number such that $S(\langle[\dots], v'_n, v_{n-1:1}\rangle) = S(\langle v'_n, [\dots], v_{n-1:1}\rangle)$. We prune all suborders $\overleftarrow{v}_{n:1}$ where $v_n < v'_n$.*

In order to prune further, we consider the dormant nodes in more detail.

Theorem 10 *Let $\overleftarrow{v}_{n:1} \in M_n$ and v^* be an optimal order of V. Then $v^* \notin \overleftarrow{v}_{n:1}$ if for any dormant node h and $k \leq n$ we have $S(\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle) > S(\langle h, [\dots], v_{n:1}\rangle)$.*

Proof With h a single dormant node and $\boldsymbol{h} = V \setminus h \cup \{v_j\}_{j=1}^n$ the set of other dormant nodes, we show that the maximal score of an order in $\langle[\boldsymbol{h}], v_n, h, v_{n-1:1}\rangle$ is greater than the maximal score in $\langle[h, \boldsymbol{h}], v_{n:1}\rangle$. From the setting we have

$$S(\langle h, [\dots], v_{n:1}\rangle) = S(\langle h, [\dots]\rangle) + S(\overleftarrow{v}_{n:1}) < S(\langle[\dots], v_{n-k}, h, v_{k-1:1}\rangle). \tag{5}$$

The maximal score of an order in $\overleftarrow{v}_{n:1}$ is

$$\begin{aligned}
&\max_{v \in \langle[\dots], v_{n:1}\rangle} S(v) = s_{\max}(\{\boldsymbol{h}, h\}|v_{n:1}) + S(\overleftarrow{v}_{n:1}) \leq s(h|\boldsymbol{h}, v_{n:1}) + s_{\max}(\{\boldsymbol{h}\}|h, v_{n:1}) + S(\overleftarrow{v}_{n:1}) \text{ / by (4)} \\
&= s_{\max}(\{\boldsymbol{h}\}|h, v_{n:1}) + S(\langle h, [\dots], v_{n:1}\rangle) < s_{\max}(\{\boldsymbol{h}\}|h, v_{n:1}) + S(\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle) \text{ / by (5)} \\
&= \max_{v \in \langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle} S(v),
\end{aligned}$$

which is the maximal score of an order in $\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle$, so $v^* \notin \overleftarrow{v}_{n:1}$. ■

Pruning 5 (Removing dormant right gaps) *We prune $\overleftarrow{v}_{n:1}$ if for any dormant node h, and for some $k \leq n$, we have $S(\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle) > S(\langle h, [\dots], v_{n:1}\rangle)$.*

In other words, we check if v_n is pushed as far to the right as possible, or if the total score could be improved by inserting another node (h) as a potential parent. If so, we prune. If they have equal score, we impose the following sorting criteria.

Pruning 6 (Ordering dormant right gaps) *We prune $\overleftarrow{v}_{n:1}$ if there is a dormant node $h > v_n$ such that $S(\langle[\dots], v_n, h, v_{n-1:1}\rangle) = S(\langle h, [\dots], v_{n:1}\rangle)$.*

3.2. Global pruning

Along with the pruning rules for right (left) orders, we have the global suborder pruning (Pruning 1) which we restate for potentially differing numbers of suborders:

Algorithm 1: Main algorithm

```
M_p ← {}
for n = 1 to p do
    M_n ← {}
    if n = 1 then
        for j ∈ V do
            m̄ ← ⟨[...], j⟩
            if hasGap(m̄) then
                continue
            end
            Update insertion scores for m̄                    /* O(p^2) */
            M_n ← M_n ∪ {m̄}
        end
    else
        M_n ← mainTransition(M_{n-1})
    end
end
return M_p
```

Pruning 7 (Removing duplicates) *At stage n, we have N orders, M_n^N, of length n. We go through all the orders with the same set of nodes and keep only one with the highest score, or in numerical order with the same score.*

One can also consider left orders of length n as subject for pruning, which we discuss in the Supplementary Material (Section S.1), where we also discuss path-search pruning (Section S.2) and detail the computational complexity of the scheme (Section S.3). We outline the main order pruning approach in Algorithm 1, while the pseudocode for the detailed steps and each Pruning is detailed in the Supplementary Material (Section S.6).

3.3. Divide-and-conquer

The result for matchings (Section S.4) suggests that if the best network is disconnected we could leverage this to further speed up the causal discovery. To extend pairwise disconnection to general networks we first consider how each parent affects the scores for each child.

We define matrices H, whose entries for node v_i are the biggest increase in score from including v_j as a parent of v_i, compared to excluding it, for any possible parent set:

$$H_{i,j}^{\max} = \max_{\mathbf{Pa}_{v_i}} \{\sigma(v_i, \mathbf{Pa}_{v_i} \cup v_j \mid D) - \sigma(v_i, \mathbf{Pa}_{v_i} \setminus v_j \mid D)\} \tag{6}$$

and the biggest decrease in score (most negative change) when including v_j as a parent of v_i:

$$H_{i,j}^{\min} = \min_{\mathbf{Pa}_{v_i}} \{\sigma(v_i, \mathbf{Pa}_{v_i} \cup v_j \mid D) - \sigma(v_i, \mathbf{Pa}_{v_i} \setminus v_j \mid D)\} \tag{7}$$

The next theorem states that if all parent sets of any node $v_i \in A \subset V$ get higher (or equal) scores when excluding any node $v_j \in B \subset V$, and vice versa, then A and B must be disconnected in the optimal DAG.

Theorem 11 If *for any subsets* $A, B \subset V$ *we have* $H^{\max}_{A,B} \leq 0$ *and* $H^{\max}_{B,A} \leq 0$*, then* A, B *are disconnected in an optimal DAG.*

Proof Assume there is an edge from $a \in A$ to $b \in B$ in the optimal DAG, $\mathcal{G}$. Since $H^{\max}_{b,a} \leq 0$, removing that edge would not decrease the score, so either $\mathcal{G}$ was not optimal which is a contradiction, or there is another optimal network without the connection. Likewise, if there is an edge from $b \in B$ to $a \in A$. ■

Corollary 12 *For disconnected components, denoted* $\mathcal{C}_u$*, of the graph induced by* $H^{\max} > 0$*, we can run the order search independently.*

This allows us to separate the network into sets of corresponding random variables which are independent. However, it may not find all disconnected components in the optimal network, so to proceed we also consider $H^{\min}$. If $A \subset V$ and $B \subset V$ are in different components of an optimal DAG, we know that adding any edge from $b \in B$ to $a \in A$ would reduce (or leave equal) the score of the parent set of a, so $H^{\min}_{a,b} \leq 0$. Likewise for edges from A to B. This argument leads to:

Theorem 13 If *subsets* $A, B \subset V$ *are disconnected in an optimal DAG* then $\forall a \in A, b \in B$*, we have* $H^{\min}_{a,b} \leq 0$ *and* $H^{\min}_{b,a} \leq 0$.

Disconnected subcomponents from the matrix $H^{\min}$ might not be disconnected in the optimal network. Instead

Corollary 14 *Disconnected subcomponents, denoted* $\mathcal{C}_l$*, in the graph induced by the boolean matrix* $H^{\min} > 0$ *are potentially disconnected in an optimal DAG.*

We are now able to formulate the following divide-and-conquer algorithm to find the optimal DAG.

Divide-and-conquer algorithm *Each subcomponent* $\mathcal{C}_l$ *from Corollary 14 is contained in one component of* $\mathcal{C}_u$ *from Corollary 12, which can be treated independently. Therefore we first organize them accordingly and then find the optimal DAG as follows:*

1. *Run the order search independently for each component* $A \in \mathcal{C}_l$ *with all other nodes as potential parents. Let this optimal ordering of* A *be denoted by* $\overleftarrow{v}_A$ *and the corresponding optimal DAG by* $\mathcal{G}_{A|V\setminus A}$.

2. *Treat the components* $\mathcal{C}_l$ *as the node set of a network* $\mathcal{N}$*. For any pair of nodes* A, B *in* $\mathcal{N}$*, draw an edge* $A \leftarrow B$ *if any node in* $\mathcal{G}_{A|V\setminus A}$ *has a parent in* B.

3. *Find the cycles in* $\mathcal{N}$*. Merge nodes along cycles and intersecting cycles to form a new component set* $\mathcal{C}_l$*. If no cycles were found go to step 4, otherwise return to step 1 and rerun the order search for modified components.*

4. *An OO is obtained by concatenating each optimal order* $\overleftarrow{v}_A$ *corresponding to the nodes* A *in* $\mathcal{N}$ *according to a topological order of* $\mathcal{N}$*. The union of the DAGs* $\mathcal{G}_{A|V\setminus A}$ *gives the corresponding optimal DAG.*

4. Polynomial causal discovery

Here we use components of the pruning rules above to prove polynomial exact causal discovery for certain classes of causal Bayesian networks. We note that this is distinct from previous results which find the optimal network within a restricted class (Chow and Liu, 1968; Gaspers et al., 2015; Grüttemeier and Komusiewicz, 2022) in polynomial time but do not prove the network is optimal overall. Instead, we are guaranteed to find the optimal network in our exact discovery and we may do so in polynomial time if the optimum satisfies certain criteria.

For example, when the empty graph is optimal, we trivially have that each order has the same score, and the score of each node does not depend on the placement of the others. Since we use the ordering restrictions, for example in Pruning 3 and 6, then we would prune every order apart from the numerically ordered one and have a single particle at each iteration. Including the computational complexity of each iteration (as detailed in Section S.3), overall this leads to a cubic algorithm, once we have the score tables. This result is however subsumed by our polynomial proof for matchings (detailed in Section S.4) and the following.

For sparse networks, our divide-and-conquer approach to finding disconnected components will lower the complexity to that of the largest component. For example, if we consider graphs with a maximum number of edges E, the largest component is bound by E. Since the number of components is bound by p, the number of merging potential components and re-running the optimal search is also at most p, following the divide-and-conquer algorithm. For each component, finding an optimal network is exponential in E in the worst case, but this does not depend on p, and hence is not exponential in p. Overall, the exact causal discovery remains polynomial in p.

More generally, for many random sparse networks, like Erdös–Rényi graphs with a density $\lesssim 1$, there is a high probability there is no giant component and that the largest components are of order $\log(p)$ (Molloy and Reed, 1995; Newman et al., 2001). Finding the components involves at most linear (in p) network searches, while each search is now exponential in $\log(p)$, and hence polynomial in p. Therefore, for any sparse network, as long as the largest connected component remains logarithmic in p we have a polynomial algorithm for exact causal discovery.

These subsets of causal Bayesian networks then constitute an exception from the general NP-hardness of exact causal discovery (Chickering, 1996; Chickering et al., 2004).

5. Simulation studies

To examine the computational complexity in practice, we performed simulation studies for networks of sizes $p \in \{10, \ldots, 30\}$ with a sample size of 300. For each network size, random graphs were generated using the default `randDAG` function from the `R`-package **pcalg** (Kalisch et al., 2012) with average density (neighbourhood size) of $d \in \{0, 0.1, \ldots, 2\}$. Continuous data were generated from a linear Gaussian structural equation model, where strengths of the edges (regression coefficients) were sampled uniformly in the range [0.25, 1]. This process was repeated with 1000 different seeds.

As a preliminary step to run our algorithm, we first calculated the score tables using the BGe score (Geiger and Heckerman, 2002; Kuipers et al., 2014) with mean hyper-parameter $\alpha_\mu = 0.1$. We estimated a DAG search space using an iterative MCMC procedure (Kuipers et al., 2022), which includes the skeleton from constraint-based testing and of the estimated DAG, and where each node is allowed an additional parent outside that core set. The procedure used the `R`-package **BiDAG** (Suter et al., 2023) and exported the output to our `C++` program (available at `https://github.com/felixleopoldo/dncDagger`). To track the complexity, we record the number of suborders

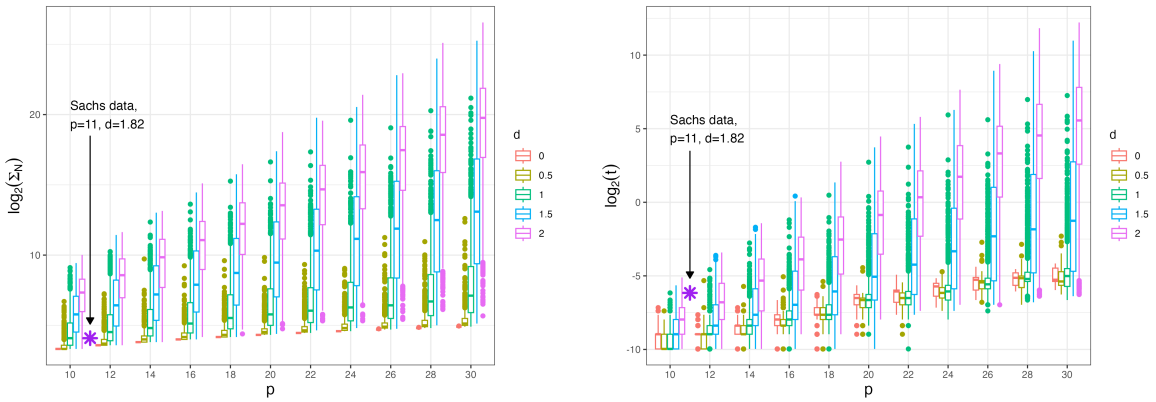


Figure 2: The logarithm of the total number of suborders Σ_N and the overall computational runtime t (in seconds) as a function of the network size p and stratified by the network density d. We additionally overlay results for the often-studied Sachs dataset (Sachs et al., 2005).

remaining after pruning, N_n, at each stage of the main algorithm (Algorithm 1) and the total runtime. We also compute the total number of suborders $\Sigma_N = \sum_n N_n$. Since this total is bound by 2^p, this is the main component where we expect exponential complexity.

The results (Figure 2 and Supplementary Figure 1) show up to linear behaviour in the growth of the logarithm of the total number of suborders for increasing network dimension, but also that it depends heavily on the network density. In fact for $d = 0$ and no edges, when the empty graph has the highest score, our ordering constraint means that we only keep one suborder at each stage so $\Sigma_N = p$, and we get sub-linear behaviour with the logarithm. For $d = 0.5$ the majority of cases also have a low number of particles, as we typically expect small components at this density, but some outliers requiring significantly more particles start to appear.

To examine this more carefully, we fit a regression line for the total number of suborders: $\log_2(\Sigma_N) = ap + b\log_2(p) + c$ for each density. The fitted coefficients (Figure 3) show that we have no exponential term ($a = 0$) up to a density of around 0.4 (and a linear term, $b = 1$), and then the logarithm of the base of the exponential term increases slowly up to a density of around 1.2 (in line with the increasing spread of outliers in Figure 2) and then more linearly for higher densities, while the polynomial term decreases to around $b = 0$. For the densities up to 2 considered, the complexity remains substantially below the theoretical maximum of $a = 1$ (and $b = 0$, when all possible 2^p suborders are kept). The fact that most examples require fewer particles at lower density is clearer when we perform a median regression (Supplementary Figure 2) with lower exponential term a.

In terms of runtime (Figure 2), the majority of cases even with $d = 1$ complete very quickly, though some examples take much longer, and the typical time also increases at the higher densities. For the runtime per suborder (Supplementary Figure 3), we would expect up to quadratic behaviour in p according to the complexity analysis, the actual time seems more linear for higher densities suggesting the complexity may be sub-quadratic in practice. Taken together, the practical runtime is much lower than the worst case $\mathcal{O}(p^2 2^p)$. As this is the best case without pruning, our pruning rules do seem to allow us to significantly lower the complexity of exact causal discovery.

As a simple example, we consider the infamous Sachs dataset (Sachs et al., 2005) which measured the levels of 11 phosphoproteins and phospholipids in individual T-cells. We use the second control

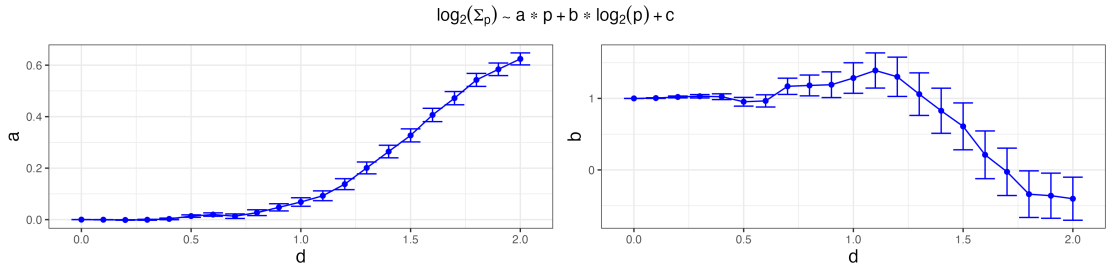


Figure 3: The regression coefficients of the logarithm of the total number of suborders $\log_2(\Sigma_N)$ against p and $\log_2(p)$, with standard errors, as a function of the network density d (the expected neighbourhood size).

dataset, log-transformed, for which most causal discovery algorithms return the same network (Rios et al., 2021). We find the optimum by only needing to track a very small number of suborders (17, annotation in Figure 2), as the network is disconnected (Supplementary Figure 4). Despite the low complexity of exact discovery in this case, the actual runtime is in line with networks of similar size and density due to overheads in the implementation, indicating room for further improvements.

Since we are considering the task of exact causal discovery, where we can prove the result is optimal (as discussed in the Introduction), we additionally compared to the ILP approach of GOBNILP (Bartlett and Cussens, 2017; Cussens et al., 2017; Cussens, 2020) by passing it the tables of scores from the same search space. Though GOBNILP uses highly optimised solvers, it only has a runtime advantage at the higher density (Supplementary Figure 5), while in line with Figure 3, our polynomial and then lower exponential complexity at lower densities provides us with better runtimes for sparser networks.

6. Discussion

We present an algorithm for exact causal discovery which improves on dynamic programming methods (Koivisto and Sood, 2004; Silander et al., 2006) by pruning orders which cannot possibly lead to the global optimum, and which expands upon previous pruning approaches (van Beek and Hoffmann, 2015). We also include pruning rules similar to the A* search approach of Yuan and Malone (2013), which finds the shortest (least costly in terms of negative score) path through the ordered power set of node orders (Figure 1) and prunes the power set by removing paths that cannot possibly improve the score. While they rely on various heuristics to prune, based on scores with potential cycles and then on avoiding cycles, we focussed on bounds at lower computational cost. When the optimal solution is a matching, our algorithm only needs to keep track of one suborder at each stage and can exit immediately, so the total complexity is polynomial. While finding the best matching amongst matchings is well-known to be polynomial, as is finding the best tree amongst trees (Chow and Liu, 1968), our study instead targets the exact discovery of finding the best network amongst all DAGs without restrictions on the type of network. Our algorithm not only completes in polynomial time when the optimum is a matching, but additionally proves the global optimality of the solution amongst all networks. We further extended our approach with a novel divide-and-conquer algorithm to detect disconnected components in the optimal DAG in an effective manner. For sparse networks with a logarithmic largest component, we can also prove optimality in polynomial time.

These results constitute a remarkable exception to the general NP-hardness of causal discovery, even with a search space with limited in-degree (Chickering, 1996; Chickering et al., 2004).

We design our pruning rules with no complexity overhead so that our worst-case complexity is the current best case (without pruning). Simulation studies show that the actual runtime and complexity can be much lower. In particular, beyond the polynomial behaviour for sparse networks, we find a slowly and then roughly linearly increasing trend between the logarithm of the base of the exponential term in the complexity and the density of the simulated networks. In our simulations, we start seeing some outliers with exponential runtime at a density of 0.5. Aymptotically we expect a sharp phase transition for random graphs with almost surely no giant component below a critical value (a density of 1 in our case), and almost certainly a giant component above. Likewise, we would expect a sharp transition from polynomial to exponential complexity asymptotically at this critical value. In practice, and away from asymptotics, our approach scales well for larger sparser networks.

We worked on the search space used by an iterative hybrid scheme (Kuipers et al., 2022) where each node may have parents from within preselected sets, plus an additional one outside that set. More typical for sampling had been to consider all possible parent sets up to a maximum in-degree (Friedman and Koller, 2003; Kuipers and Moffa, 2017), or for discovery to prune further to tables of candidate parent sets (Friedman et al., 1999; Teyssier and Koller, 2005). This is the standard input to current state-of-the-art exact causal discovery algorithms (Malone et al., 2018). The hybrid setting offers several advantages. First, it typically offers good performance in causal discovery (Rios et al., 2021). Second, if the initial search space is found with a consistent method, like the pc-algorithm, and if the score function is consistent, like with a penalised likelihood, then we retain consistency in our exact causal discovery. Third, the hybrid approach allows us to precompute all scores needed for the search, rather than potentially needing to look through tables of scores of candidate parent sets. As such we can remove this factor from influencing the algorithm runtime. In particular, we observe clear trends of the runtime as a function of the density of the network and its size, which seems much more predictable than for other exact solvers (Malone et al., 2018). We can therefore examine the hardness of exact causal discovery more precisely.

At lower densities, our order-based pruning algorithm is generally faster than the ILP approach of GOBNILP (Bartlett and Cussens, 2017; Cussens et al., 2017; Cussens, 2020), which was also typically found to be the fastest compared to other exact algorithms (Malone et al., 2018), especially for smaller and sparser instances (Trösser et al., 2021). Our new divide-and-conquer and pruning rules might help improve ILP approaches, while these may suggest other avenues for better pruning our search. Overall, our approach offers a useful framework for further lowering the complexity of exact causal discovery and extending its use for larger and sparser networks.

References

Mark Bartlett and James Cussens. Integer Linear Programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

David M. Chickering. Learning Bayesian networks is NP-complete. *Learning from data: Artificial intelligence and statistics V*, pages 121–130, 1996.

David M. Chickering, David Heckerman, and Chris Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14:462–467, 1968.

James Cussens. GOBNILP: Learning Bayesian network structure with integer programming. In *International Conference on Probabilistic Graphical Models*, pages 605–608, 2020.

James Cussens, Matti Järvisalo, Janne H. Korhonen, and Mark Bartlett. Bayesian network structure learning with integer programming: Polytopes, facets and complexity. *Journal of Artificial Intelligence Research*, 58:185–229, 2017.

Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–125, 2003.

Nir Friedman, Iftach Nachman, and Dana Pe'er. Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

Serge Gaspers, Mikko Koivisto, Mathieu Liedloff, Sebastian Ordyniak, and Stefan Szeider. On finding optimal polytrees. *Theoretical Computer Science*, 592:49–58, 2015.

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

Niels Grüttemeier and Christian Komusiewicz. Learning Bayesian networks under sparsity constraints: A parameterized complexity analysis. *Journal of Artificial Intelligence Research*, 74: 1225–1267, 2022.

David Heckerman and Dan Geiger. Learning Bayesian networks: A unification for discrete and Gaussian domains. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 274–284, 1995.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47:1–26, 2012.

Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

Jack Kuipers and Giusi Moffa. Partition MCMC for inference on acyclic digraphs. *Journal of the American Statistical Association*, 112:282–299, 2017.

Jack Kuipers, Giusi Moffa, and David Heckerman. Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics*, 42:1689–1691, 2014.

Jack Kuipers, Polina Suter, and Giusi Moffa. Efficient sampling and structure learning of Bayesian networks. *Journal of Computational and Graphical Statistics*, 31:639–650, 2022.

Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Empirical hardness of finding optimal Bayesian network structures: Algorithm selection and runtime prediction. *Machine Learning*, 107:247–283, 2018.

Ehsan Mokhtarian, Mohmmadsadegh Khorasani, Jalal Etesami, and Negar Kiyavash. Novel ordering-based approaches for causal structure learning in the presence of unobserved variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12260–12268, 2023.

Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6:161–180, 1995.

Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24:227–234, 1995.

Mark E.J. Newman, Steven H. Strogatz, and Duncan J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64:026118, 2001.

Robert C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36:1389–1401, 1957.

Felix L. Rios, Giusi Moffa, and Jack Kuipers. Benchpress: A scalable and versatile workflow for benchmarking structure learning algorithms for graphical models. *arXiv:2107.03863*, 2021.

Karen Sachs, Omar Perez, Dana Pe'er, Douglas A. Lauffenburger, and Garry P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308:523–529, 2005.

Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, 115:235–253, 2019.

Tomi Silander, Petri Kontkanen, and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 445–452, 2006.

Polina Suter, Jack Kuipers, Giusi Moffa, and Niko Beerenwinkel. Bayesian structure learning and sampling of Bayesian networks with the R package BiDAG. *Journal of Statistical Software*, 105: 1–31, 2023.

Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 584–590, 2005.

Fulya Trösser, Simon de Givry, and George Katsirelos. Improved acyclicity reasoning for Bayesian network structure learning with constraint programming. *arXiv:2106.12269*, 2021.

Peter van Beek and Hella-Franziska Hoffmann. Machine learning of Bayesian networks using constraint programming. In *Principles and Practice of Constraint Programming: 21st International Conference*, pages 429–445, 2015.

Changhe Yuan and Brandon Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

Appendix S. Supplementary Material

S.1. Pruning left orders

Here we consider left orders of length n as subject for pruning. The simplest strategy is again when the last node added could fit better earlier in the order.

Pruning 8 (Optimal back) *We prune* $\overrightarrow{v}_{1:n}$ *if for some* $k < n$ *we have* $S(\langle v_{1:k-1}, v_n, v_{k:n-1}, [\dots]\rangle) > S(\overrightarrow{v}_{1:n})$.

Analogously to Pruning 3, we ensure the two nodes in the back are in numerical order if they can be swapped without affecting the score.

Pruning 9 (Ordered back) *We prune* $\overrightarrow{v}_{1:n}$ *if* $v_n < v_{n-1}$ *and* $S(\overrightarrow{v}_{1:n}) = S(\langle v_{1:n-2}, v_n, v_{n-1}, [\dots]\rangle)$.

We also consider the analogue of Theorem 9 for left orders when appending new nodes to the back.

Theorem 15 *If* $S(\overrightarrow{v}_{1:n}) = S(\langle v_{1:n-1}, [\dots], v_n\rangle)$, *then* $\max_{v\in\langle v_{1:n-1},h,[\dots]\rangle} S(v) \geq \max_{v\in\langle v_{1:n},h,[\dots]\rangle} S(v)$ *for any dormant node* h.

Proof The equality means that the score of v_n is independent of the other nodes. However, the score of some of the dormant nodes h may depend on v_n. In particular for any h we have that

$$\max_{v\in\langle v_{1:n},h,[\dots]\rangle} S(v) \leq \max_{v\in\langle v_{1:n-1},h,v_n,[\dots]\rangle} S(v) \leq \max_{v\in\langle v_{1:n-1},h,[\dots]\rangle} S(v).$$

■

Theorem 15 implies that, with equal scores, as long as there is a dormant node with a lower number than v_n then it is not necessary to keep v_n at stage n, as it could be added later equally well or better.

Pruning 10 (No left gaps) *We prune* $\overrightarrow{v}_{1:n}$ *if* $S(\overrightarrow{v}_{1:n}) = S(\langle v_{1:n-1}, [\dots], v_n\rangle)$ *and for any dormant node* h *we have* $h < v_n$.

Theorem 15 is the analogue of Theorem 9 and allows us to prune for the left orders similarly to the right orders in Pruning 4. Since scores depend on possible parents, however, the pruning conditions differ due to this asymmetry, and the analogue of Theorem 10 does not seem to immediately lead to another pruning rule.

S.2. Path-search pruning

In the A* search algorithm of Yuan and Malone (2013), suborders are pruned when an upper bound of their scores including the dormant nodes is less than (or equal) to the current best order score. We will denote the current best (full) order v^* with score $S^* = S(v^*)$, and $g(\overleftarrow{v}_{n:1})$ as the upper bound score of the dormant nodes of $\overleftarrow{v}_{n:1}$.

Pruning 11 (A*) *We prune* $\overleftarrow{v}_{n:1}$ *if* $S(\overleftarrow{v}_{n:1}) + g(\overleftarrow{v}_{n:1}) \leq S^*$.

The task then is to find as low an upper bound as possible in low computational time. To proceed we use the observation (Lemma 8 and Yuan and Malone (2013)) that for any partition Λ of the dormant nodes $U = V \setminus \{v_j\}_{j=1}^n$ in a right suborder $\overleftarrow{v}_{n:1}$ we have

$$s_{\max}(\{U\}|V \setminus U) \leq \sum_{i=1}^{m} s_{\max}(\{\Lambda_i\}|V \setminus \Lambda_i) \tag{8}$$

where the Λ_i are the m parts of Λ. In Yuan and Malone (2013) they considered separating into sets of size k and finding the optimal sets that minimise the right hand side above. For $k = 2$ this is equivalent to finding optimal matchings, which can be achieved in cubic time with a variant of Edmond's blossom algorithm. However, for $k > 2$ the problem is computationally hard and in Yuan and Malone (2013) they focussed on heuristic approaches to find bounds.

Along with pruning, we can also update our current best order v^* if we can find a better one. Here, guided by the complexity, we follow a different heuristic approach for the upper bound, that will later allow us to prove optimality in polynomial time for matchings. We consider the best scoring tree on the dormant nodes, which we denote $\mathcal{T}(\overleftarrow{v}_{n:1})$. To obtain this tree we define the matrix F with entries

$$F_{ij} = S(\langle[\ldots], v_i, v_j\rangle) - S(\langle[\ldots], v_i\rangle) - S(\langle[\ldots], v_j\rangle) \tag{9}$$

which is the difference in score between allowing an edge between nodes v_i and v_j and excluding such an edge. The matrix F is symmetric, since pairwise edges are score-equivalent, and the minimum value will be 0 in the case excluding the edge fits the data better.

The highest score of a tree network on the dormant nodes $U = V \setminus \{v_j\}_{j=1}^n$ of a right order $\overleftarrow{v}_{n:1}$ is

$$f(\overleftarrow{v}_{n:1}) = \sum_{e \in \mathcal{T}(\overleftarrow{v}_{n:1})} F_e + \sum_{u \in U} S(\langle[\ldots], u\rangle) \tag{10}$$

where e are the edges in the undirected maximum spanning tree $\mathcal{T}(\overleftarrow{v}_{n:1})$ found from the relevant submatrix of F, for example, with Prim's algorithm (Prim, 1957). Since trees are score-equivalent any direction imposed on the tree and, more importantly, any ordering of the dormant nodes compatible with the best scoring tree will have an equal or higher score than $f(\overleftarrow{v}_{n:1})$.

Update 1 *We update v^* if for any right order $\overleftarrow{v}_{n:1}$*

$$f(\overleftarrow{v}_{n:1}) + S(\langle[\ldots], \overleftarrow{v}_{n:1}\rangle) > S^* \tag{11}$$

In addition we can also compare to the current best order

Update 2 *We update v^* if for any right order $\overleftarrow{v}_{n:1}$*

$$S(\langle v^* \setminus \overleftarrow{v}_{n:1}, \overleftarrow{v}_{n:1}\rangle) > S^* \tag{12}$$

where $v^* \setminus \overleftarrow{v}_{n:1}$ denotes removing the elements of $\overleftarrow{v}_{n:1}$ from v^* and keeping the rest in the same order.

For the upper bound instead, given a tree we can find the optimal matching in linear time. We then use this matching for our upper bound $g(\overleftarrow{v}_{n:1})$. For the computation we use the symmetric matrix G which records the maximal relative score of any pair of nodes

$$G_{ij} = s_{\max}(\{v_i, v_j\}|V \setminus \{v_i, v_j\}) - S(\langle v_i, [\dots]\rangle) - S(\langle v_j, [\dots]\rangle) \tag{13}$$

If we define $\mathcal{M}(\overleftarrow{v}_{n:1})$ to be the optimal matching obtained from $\mathcal{T}(\overleftarrow{v}_{n:1})$, the upper bound is

$$g(\overleftarrow{v}_{n:1}) = \sum_{e \in \mathcal{M}(\overleftarrow{v}_{n:1})} G_e + \sum_{u \in U} S(\langle u, [\dots]\rangle) \tag{14}$$

If the best tree score matches the upper bound $g(\overleftarrow{v}_{n:1})$, the tree network on the dormant nodes is optimal and we do not need to continue expanding the suborder:

Pruning 12 (Optimality) *If $f(\overleftarrow{v}_{n:1}) = g(\overleftarrow{v}_{n:1})$, we prune $\overleftarrow{v}_{n:1}$.*

A neat corollary is that, if the best network is a matching, we can prove optimality in quadratic time (see Section S.4). Finally, if the actual score of the dormant nodes within any of the update rules is larger than $f(\overleftarrow{v}_{n:1})$ but matches the upper bound $g(\overleftarrow{v}_{n:1})$, we can also prune.

S.3. Computational complexity

For the complexity considerations, each score computation for a single node can be up to $\mathcal{O}(p)$ in our setting. To aid the computations, for each suborder $\overleftarrow{v}_{n:1}$ we keep track of the highest score of the remaining dormant nodes $h \in V \setminus \{v_j\}_{j=1}^n$ in the front, further along the order, or completely at the front with a gap: $S(\langle[\dots], h, v_{n:1}\rangle)$, $\max_{k=1\dots n-1} S(\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle)$ and $S(\langle h, [\dots], v_{n:1}\rangle)$. Each suborder at stage n is then stored as the ordered list of the n nodes already included and the table of $3(p-n)$ scores.

After stage $(n-1)$, say we have N_{n-1} suborders $O_{n-1}^{N_{n-1}}$ of length $(n-1)$. For each of these we try to append the remaining $(p-n+1)$ nodes to the front for Pruning 2 and 3. With the tabulated scores, the optimal front condition can be checked in $\mathcal{O}(1)$. Since $N_{n-1} \leq \binom{p}{n-1}$ the total complexity of this pruning is bound by $\mathcal{O}(p2^p)$.

For removing right gaps with Pruning 4, we can check the condition in $\mathcal{O}(p)$ from the tabulated scores and already prune some of the $(p-n+1)$ nodes which could be added to the front. The total complexity is likewise bound by $\mathcal{O}(p2^p)$.

The more complicated task is the removing dormant right gaps with Pruning 5 and 6. Here we need the score for each potential dormant node of which there are $\mathcal{O}(p)$ and each new score computation is also $\mathcal{O}(p)$ in our setting. With the $\mathcal{O}(p)$ nodes we try to append to the front this could naïvely mean a complexity of $\mathcal{O}(p^3 N_{n-1})$, however we can avoid this overhead and reduce the complexity by reorganising our algorithm and first running the global pruning, as follows.

For the global Pruning 7, after the optimal front and right gap pruning we have $N \leq (p-n+1)N_{n-1}$ potential suborders. Of these we remove any duplicates with the same node set, keeping the highest scoring or in numerical order. To compare the sets is $\mathcal{O}(pN)$ since we also need to check the suborder set elements. By summing over n the total complexity of this step is bound by $\mathcal{O}(p^2 2^p)$. Note that for the equivalent pruning in Section 2.3 the complexity was only $\mathcal{O}(p2^p)$ since we already know the suborder set elements from the Hasse diagram.

After the global pruning we have $N' \leq \binom{p}{n}$ suborders remaining. For each of these we now update their insertion score tables. For each suborder $\langle[\dots], v_{n:1}\rangle$ of length n, for each of the dormant nodes $h \in V \setminus \{v_j\}_{j=1}^n$ we compute $S(\langle[\dots], h, v_{n:1}\rangle)$, $S(\langle h, [\dots], v_{n:1}\rangle)$ and $S(\langle[\dots], v_n, h, v_{n-1:1}\rangle)$. Since these are all updates of one node compared to the scores previously computed for $\langle[\dots], v_{n-1:1}\rangle$, each one takes $\mathcal{O}(p)$ and the whole process takes $\mathcal{O}(p^2)$. For the maximum score

$$\max_{k=1\dots n-1} S(\langle[\dots], v_{n:k}, h, v_{k-1:1}\rangle) = s(v_n|h, v_{n-1:1}) + \max_{k=1\dots n-1} S(\langle[\dots], v_{n-1:k}, h, v_{k-1:1}\rangle) \tag{15}$$

the maximum on the second line is computed by taking the largest score with the dormant node at the front or further along the order previously computed for $\overleftarrow{v}_{n-1:1}$ and hence is $\mathcal{O}(1)$. Updating the score tables therefore is $\mathcal{O}(p^2 N')$ and the total complexity is again bound by $\mathcal{O}(p^2 2^p)$.

With the updated score tables for the suborders of length n, checking the condition for the removal of dormant right gaps with Pruning 5 and 6 takes $\mathcal{O}(1)$ per dormant node and hence $\mathcal{O}(p)$ for each suborder. The total complexity is then bound by $\mathcal{O}(p2^p)$. In particular we can check the gap condition as we are updating the score tables.

For pruning left suborders, the total complexity of Pruning 8 and 9 is, analogously for the right suborders, bound by $\mathcal{O}(p2^p)$. For removing left gaps with Pruning 10 we can check the condition in $\mathcal{O}(1)$ for each potential new addition at the back by looking up the relevant insertion scores. To check the numerical ordering condition we should additionally track the lowest number of the nodes outside the suborder to avoid any extra computational costs. As for the optimal back pruning, the total complexity is bound by $\mathcal{O}(p2^p)$.

For the path-search Pruning 11, computing the lower bound with Prim's algorithm (Prim, 1957) for undirected maximum spanning trees, and also extracting a matching for the upper bound for Pruning 12, is $\mathcal{O}(p^2)$. For each update rule for the best suborder, we need to score specific left suborders of the dormant nodes, again with complexity $\mathcal{O}(p^2)$. The total complexity of the path-search based pruning and updates is therefore bound by $\mathcal{O}(p^2 2^p)$.

The overall worst case total complexity of $\mathcal{O}(p^2 2^p)$ is the same as the baseline complexity of Section 2.3. When including the divide-and-conquer rules, we run the above scheme on each potentially disconnected component, but after discovering the optimal network on each component we may need to merge components and rerun. The number of components is limited by p, and hence so is the number of times we could merge and rerun. However, the sizes of the components are also smaller than p, so for example the scheme where we add one node at a time and rerun would have complexity terms involving $\sum_{k=1}^{p} k^2 2^k$, again leading to an overall complexity of $\mathcal{O}(p^2 2^p)$.

Including all the pruning rules does not therefore worsen the complexity class of the algorithm. Because we prune suborders, however, we may be able to achieve lower complexity in practice than with the dynamic programming approach.

S.4. Polynomial discovery of matchings

For the lower bound from the maximum spanning tree in Equation (10) there may be edges with $F_{ij} = 0$. In this case, the best network would not have an edge between v_i and v_j. If the best tree is a matching, we can only have edges between disconnected pairs of nodes. If v_i and v_j are paired in the matching, then we must have $F_{ik} = 0$ for all $k \neq j$ and $F_{kj} = 0$ for all $k \neq i$. If not the MST would have placed an edge between v_i or v_j and another node. Note that this would not have created a cycle since the other node would be part of a distinct disconnected pair.

BOUNDS FOR MATCHINGS

If the best network is a matching, for each connected pair v_i and v_j we must have

$$S(\langle v_i, v_j, [\dots]\rangle) = S(\langle [\dots], v_i, v_j\rangle) \tag{16}$$

and

$$S(\langle v_j, v_i, [\dots]\rangle) = S(\langle [\dots], v_j, v_i\rangle) \tag{17}$$

since otherwise we could include other parents for one of the nodes, increase the score and no longer have a matching. Combined this is

$$s_{\max}(\{v_i, v_j\}|V \setminus \{v_i, v_j\}) = S(\langle [\dots], v_i, v_j\rangle) = S(\langle [\dots], v_j, v_i\rangle) \tag{18}$$

Looking at one of the nodes

$$S(\langle v_i, [\dots]\rangle) = S(\langle v_i, v_j, [\dots]\rangle) - S(\langle [v_i], v_j, [\dots]\rangle) = S(\langle [\dots], v_i, v_j\rangle) - S(\langle [\dots], v_j\rangle) \tag{19}$$

comparing to Equations (9) and (13) we have

$$G_{ij} = -F_{ij} \tag{20}$$

On the other hand, if v_i and v_k are disconnected in the matching,

$$s_{\max}(\{v_i, v_k\}|V \setminus \{v_i, v_k\}) = S(\langle v_i, [\dots]\rangle) + S(\langle v_k, [\dots]\rangle) \tag{21}$$

since their relative order does not matter. Therefore

$$G_{ik} = 0 = -F_{ik} \tag{22}$$

and overall $G = -F$ when the best network is a matching.

EQUALITY FOR MATCHINGS

Since G and F are opposite of each other, the minimum spanning tree for G is a maximum spanning tree for F, and so the same matching is an MST for G. As a final step, we need to show equality of the lower and upper bounds:

$$\begin{aligned} f(\overleftarrow{v}_{n:1}) &= \sum_{e \in \mathcal{M}(\overleftarrow{v}_{n:1})} F_e + \sum_{v \in V} S(\langle [\dots], v\rangle) \\ = g(\overleftarrow{v}_{n:1}) &= \sum_{e \in \mathcal{M}(\overleftarrow{v}_{n:1})} G_e + \sum_{v \in V} S(\langle v, [\dots]\rangle) \end{aligned} \tag{23}$$

Proof For any disconnected node in the matching, say v_k, including any additional parents cannot improve the score, since one could include edges to those parents and have a tree, contradicting that our best tree is a matching. For disconnected nodes we have

$$S(\langle [\dots], v_k\rangle) = S(\langle v_k, [\dots]\rangle) \tag{24}$$

and no contributions from F or G. Otherwise, for connected pairs, v_i and v_j, we have

$$\begin{aligned} & F_{ij} + S(\langle [\ldots], v_i \rangle) + S(\langle [\ldots], v_j \rangle) \\ = & S(\langle [\ldots], v_i, v_j \rangle) = S(\langle v_i, v_j, [\ldots] \rangle) \\ = & G_{ij} + S(\langle v_i, [\ldots] \rangle) + S(\langle v_j, [\ldots] \rangle) \end{aligned} \tag{25}$$

from the definition in Equations (9) and (13), and the result in Equation (16). The terms agree for disconnected nodes and pairs of connected nodes in the matching, and so agree for the entire matching. ■

POLYNOMIAL OPTIMALITY TESTING

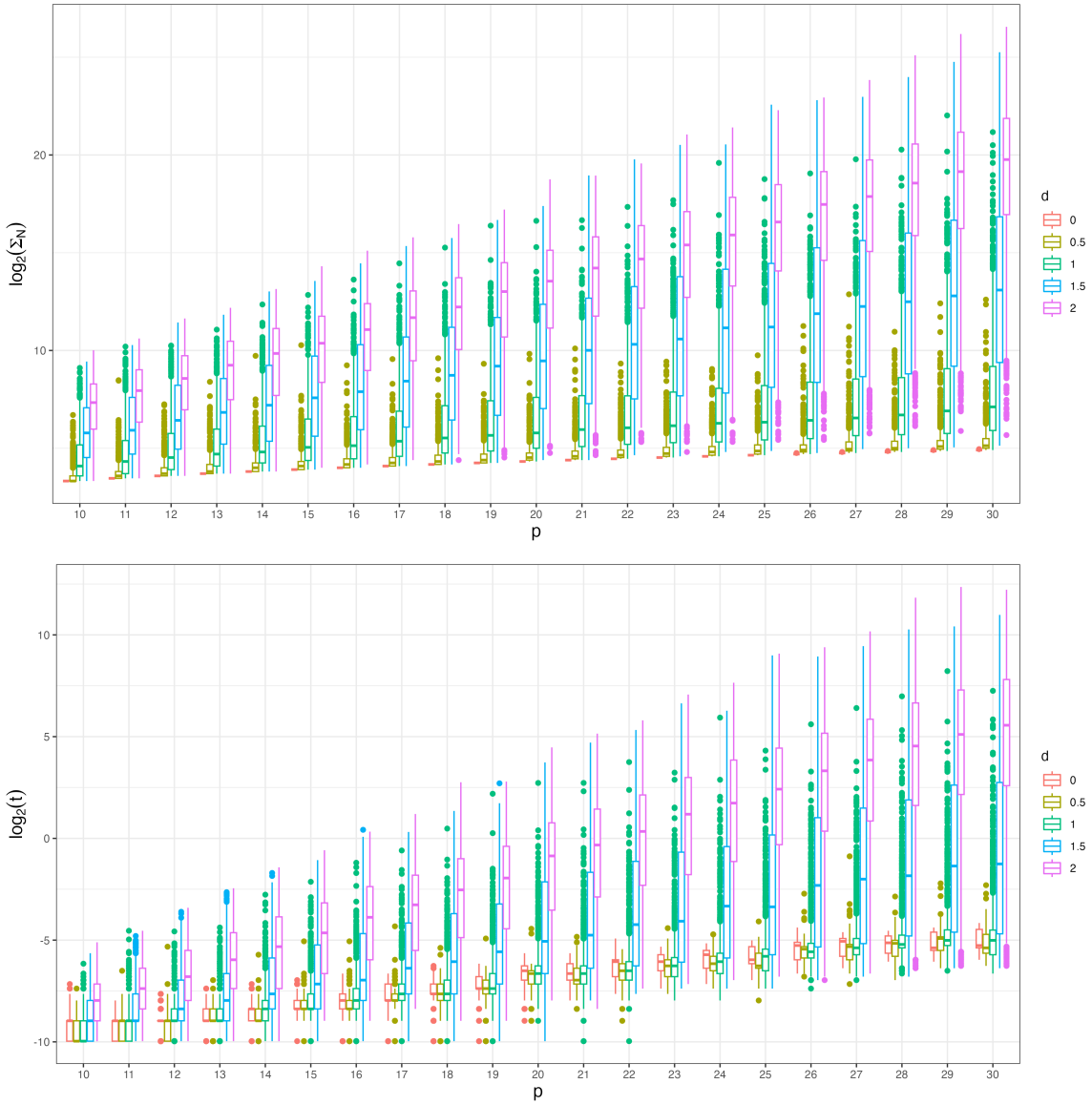
In computing the score tables for F to find the best tree, we only need to consider each node with no, or one parent, and building the matrix F can be done in quadratic time for usual scores like the BDe or BGe (Heckerman and Geiger, 1995; Geiger and Heckerman, 2002; Kuipers et al., 2014). Finding the best tree network is well-known to be polynomial (Chow and Liu, 1968), and we can do so from F in quadratic time (Prim, 1957).

For the matrix G, however, we need the best scoring parent set for each node, and the best subset selection problem is generally NP-hard (Natarajan, 1995). With a restricted search space (Kuipers et al., 2022), or a restricted in-degree (Friedman and Koller, 2003), computing the score tables for G becomes polynomial.

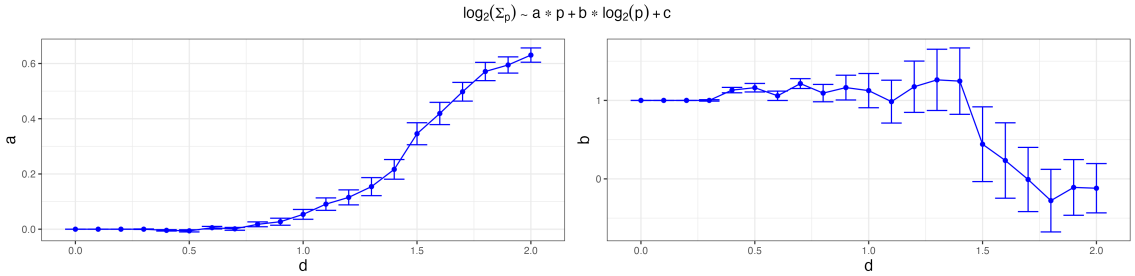
Finding the optimal DAG is NP-hard even for a restricted in-degree (of at least 2, Chickering, 1996, or at least 3, Chickering et al., 2004), so we consider that case where constructing G is still polynomial. Finding the upper bound from G is linear and can likewise be computed in polynomial time.

Overall, if the best causal Bayesian network in a search space with restricted in-degree is a matching then things simplify. We can find it (from F) and prove its optimality (from G) in polynomial time, and, once the score tables have been computed, in quadratic time.

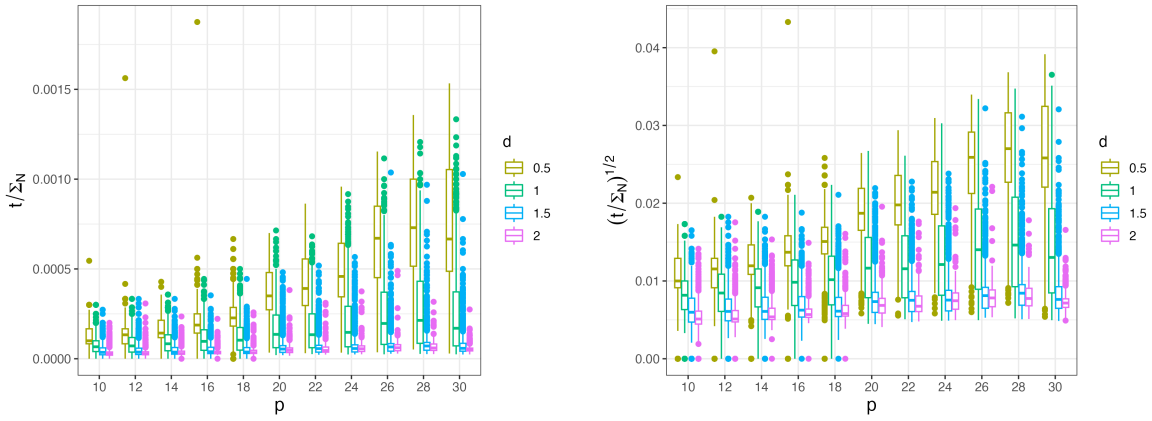
S.5. Supplementary Figures



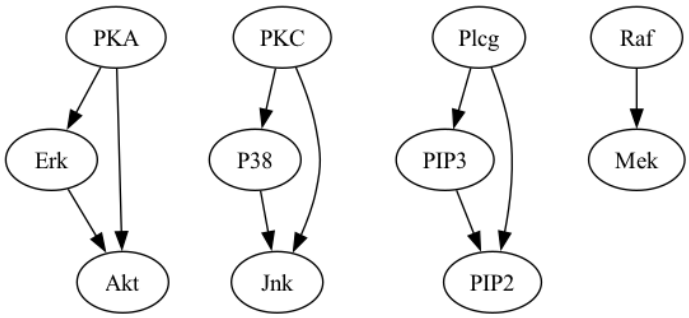
Supplementary Figure 1: More complete version of Figure 2 showing the logarithm of the total number of suborders Σ_N and runtime t (in seconds) for all the network sizes $p \in \{10, \ldots, 30\}$, stratified by the network density d.



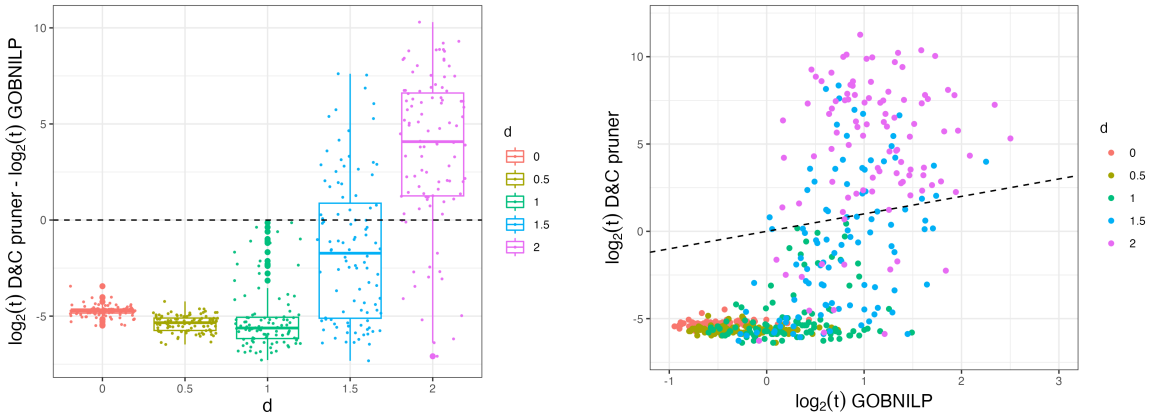
Supplementary Figure 2: The regression coefficients for a median regression of the logarithm of the total number of suborders $\log_2(\Sigma_N)$ against p and $\log_2(p)$, with standard errors, as a function of the network density d.



Supplementary Figure 3: The ratio of the total runtime to the total number of suborders: t/Σ_N for networks of different density d. We would expect up to quadratic behaviour in p, though it appears more linear for higher densities. With the square root transform (right panel), the apparent sub-linear behaviour for higher densities suggests an overall sub-quadratic dependency.



Supplementary Figure 4: The optimum network learned from the Sachs dataset (Sachs et al., 2005), which aligns with that learned with most causal discovery algorithms (Rios et al., 2021). Each component is fully connected and hence score-equivalent to any acyclic redirecting of the edges.



Supplementary Figure 5: The logarithm of the relative runtime of our approach (D&C pruner) compared to GOBNILP for networks of size $p = 30$ along with the comparison of the log runtimes (in seconds) of the two approaches, coloured by density.

S.6. Supplementary Algorithms

Algorithm 2: mainTransition

Data: M_{n-1} list of orders of length $n-1$
$M_n \leftarrow \{\}$
for $\bar{m} = \overleftarrow{v}_{n-1:1} \in M_{n-1}$ /* $\mathcal{O}(\binom{p}{n-1})$ */
do
  $R \leftarrow noRightGaps(\bar{m})$ /* $\mathcal{O}(p)$ */
  for $j \in V \setminus \{v_j\}_{j=1}^{n-1}$ /* $\mathcal{O}(p)$ */
  do
    if $j \notin R$ /* $\mathcal{O}(1)$ */
    then
      continue
    end
    if $!optimalFront(\bar{m}, j)$ /* $\mathcal{O}(1)$ */
    then
      continue
    end
    if $!orderedFront(\bar{m}, j)$ **then**
      continue
    end
    $\bar{u} \leftarrow \langle [\dots], j, v_{n-1:1} \rangle$
    Compute $S(\bar{u})$ /* $\mathcal{O}(1)$ */
    $M_n \leftarrow M_n \cup \{\bar{u}\}$
  end
end
$M_n \leftarrow pruneEqualSets(M_n)$ /* $\mathcal{O}(\binom{p}{n} n \times p)$ */
for $\bar{m} \in M_n$ /* $O(\binom{p}{n})$ */
do
  Update insertion scores for $\bar{m}$ /* $\mathcal{O}(p^2)$ */
end
for $\bar{m} \in M_n$ /* $\mathcal{O}(\binom{p}{n})$ */
do
  if $hasDormantGap(\bar{m})$ /* $\mathcal{O}(p)$ */
  then
    $M_n \leftarrow M_n \setminus \{\bar{m}\}$
  end
  if $!orderedDormantGap(\bar{m})$ /* $\mathcal{O}(p)$ */
  then
    $M_n \leftarrow M_n \setminus \{\bar{m}\}$
  end
end
return M_n

Algorithm 3: noRightGaps (*c.f.* Pruning 4)

Data: A right order $\bar{m} = \overleftarrow{v}_{n-1:1}$
$R = \{\}$
$m \leftarrow -1$
for $h \in V \setminus \{v_j\}_{j=1}^{n-1}$ **do**
  if $S(\langle h, [\dots], v_{n-1:1}\rangle) = S(\langle [\dots], h, v_{n-1:1}\rangle)$ **then**
    if $h > m$ **then**
      $m \leftarrow h$
    end
  end
end
for $h \in V \setminus \{v_j\}_{j=1}^{n-1}$ **do**
  if $h \geq m$ **then**
    $R \leftarrow R \cup \{h\}$
  end
end
return R

Algorithm 4: optimalFront (*c.f.* Pruning 2)

Data: A right order $\bar{m} = \overleftarrow{v}_{n-1:1}$
Data: A new front node j
Data: Pre-calculated max insert scores for $\bar{m}$
$m \leftarrow S(\langle [\dots], j, v_{n:1}\rangle)$
$s \leftarrow \max_{k \leq n} S(\langle [\dots], v_{n:k}, j, v_{k-1:1}\rangle)$ /* $\mathcal{O}(1)$ */
if $s > m$ **then**
  return False
end
return True

Algorithm 5: orderedFront (*c.f.* Pruning 3)

Data: A right order $\overleftarrow{v}_{n:1}$
if $S(\overleftarrow{v}_{n:1}) = S(\langle [\dots], v_{n-1}, v_n, v_{n-2:1}\rangle)$ **then**
  if $v_n < v_{n-1}$ **then**
    return True
  end
end
return False

Algorithm 6: hasDormantGap (*c.f.* Pruning 5)

Data: A right order $\bar{m} = \overleftarrow{v}_{n:1}$
Data: Pre-calculated max insert scores for $\bar{m}$.
$m \leftarrow S(\langle h, [\dots], v_{n:1} \rangle)$ /* $\mathcal{O}(1)$ */
for $h \in V \setminus \{v_j\}_{j=1}^{n}$ /* $\mathcal{O}(p)$ */
do
  $s \leftarrow \max_k S(\langle [\dots], v_{n:k}, h, v_{k-1:1} \rangle)$ /* $\mathcal{O}(1)$ */
  if $s > m$ **then**
    return True
  end
end
return False

Algorithm 7: orderedDormantGap (*c.f.* Pruning 6)

Data: A right order $\bar{m} = \overleftarrow{v}_{n:1}$
Data: Pre-calculated max insert scores for $\bar{m}$
$m \leftarrow S(\langle h, [\dots], v_{n:1} \rangle)$ /* $\mathcal{O}(1)$ */
for $h \in V \setminus \{v_j\}_{j=1}^{n}$ /* $\mathcal{O}(p)$ */
do
  $s \leftarrow S(\langle [\dots], v_n, h, v_{n-1:1} \rangle)$ /* $\mathcal{O}(1)$ */
  if $s = m$ & $h < v_n$ **then**
    return True
  end
end
return False

Algorithm 8: pruneEqualSets (*c.f.* Pruning 7)

Data: $A \in \{0,1\}^{N \times p}$ an $N \times p$ binary row matrix indicating elements in orders
Data: $order\_scores \in \mathbf{R}^N$
$start\_inds \leftarrow [1, 1, 2, \ldots, N]$ /* First element (1) indicates the column to be used in A */
$L = \{\}$ /* List of indices for the maximal scoring order of same sets */
$Q = \{\}$ /* Indices to consider */
$Q.push\_back(start\_inds)$
while $Q \neq \{\}$ **do**
  $inds \leftarrow Q.pop\_back()$
  $k \leftarrow inds[1]$
  $one\_inds \leftarrow [k+1]$ /* To be used in col $k+1$ of A */
  $zero\_inds \leftarrow [k+1]$
  for $i \in inds$ **do**
    if $A_{ik} = 0$ **then**
      $zero\_inds.push\_back(i)$
    end
    $one\_inds.push\_back(i)$
  end
  if $k = p$ /* Last column, where scores are used */
  then
    if $zero\_inds.size() \neq 0$ **then**
      $i^* \leftarrow \underset{i \in zero\_inds}{\arg\max}\ order\_scores[i]$
      $L.push\_back(i^*)$
    end
    if $one\_inds.size() \neq 0$ **then**
      $i^* \leftarrow \underset{i \in one\_inds}{\arg\max}\ order\_scores[i]$
      $L.push\_back(i^*)$
    end
  else
    if $zero\_inds.size() > 1$ **then**
      $Q.push\_back(zero\_inds)$
    end
    if $one\_inds.size() > 1$ **then**
      $Q.push\_back(one\_inds)$
    end
  end
end
return L