On Tractability of Learning Bayesian Networks with Ancestral Constraints

Juha Harviainen
University of Helsinki

Pekka Parviainen
University of Bergen

Abstract

Expert knowledge can greatly reduce the complexity of Bayesian network structure learning by constraining the search space. These constraints can come in the form of ancestral constraints that relate to the existence of paths between nodes. When the constraints are compiled into a directed acyclic graph, the complexity of learning with ancestral constraints is connected to the number of ideals of the constraint graph. First, we consider precedence constraints which define a partial order that the structure must obey. Taking the path cover number of the constraint graph as a parameter, we extend earlier results to the problems of sampling and weighted counting of network structures. We also consider the problems with related ancestral constraints which state that a node must or cannot be an ancestor of another. With positive ancestral constraints, we show that the problems are tractable under the additional assumption that the constraint graph has only a small number of incomparable edges. On the other hand, the optimization problem is NP-hard with negative ancestral constraints when the path cover number is at least two. Finally, we show that these problems become fixed-parameter tractable if the constraints are compatible with a subclass of partial orders called bucket orders.

1 INTRODUCTION

Finding a Bayesian network structure that models relationships between nodes (random variables) the best is famously an NP-hard problem (Chickering, 1995). This is partially caused by the number of directed acyclic graphs (DAGs) being super-exponential in the number of nodes. Score-based methods assign a score to each DAG describing how well it fits the assumptions and the data. The fastest exact learning algorithms take exponential time in the worst case to learn an optimal structure given a *modular* score function that decomposes the score into node-wise *local scores* (Ott et al., 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006). Although heuristics can provide significant speedups to these algorithms (Yuan et al., 2011; He et al., 2023), they are not guaranteed to perform well.

However, we often have some a priori knowledge about the structure, narrowing down the search space (see, e.g., de Campos and Ji, 2011; Parviainen and Koivisto, 2013; Li and van Beek, 2018; Peruvemba Ramaswamy and Szeider, 2022). Causal relationships from domain knowledge can be one source of such information, and another one can be time: if v happens before u, then u is not a cause of v. Here, we study a hierarchy of *ancestral constraints* that relate to the existence of paths between nodes. *Positive* and *negative ancestral constraints* state that a node must or cannot be an ancestor of another, and *precedence constraints* assert that a node precedes another in a topological order. A DAG that satisfies a set of constraints keeps satisfying them if positive ancestral constraints are replaced by precedence constraints or precedence constraints by negative ancestral constraints. We collect the constraints into a DAG called a *positive/negative ancestral constraint graph* (PACG/NACG) or *precedence constraint graph* (PCG) depending on their type, and use *ancestral constraint graph* (ACG) as an umbrella term. Arc constraints that relate to the existence of edges can be integrated directly to the local scores.

An alternative approach to structure learning with constraints is given by superstructures that encode all allowed undirected or directed dependencies between the variables (Perrier et al., 2008; Ordyniak and Szeider, 2013; Ganian and Korchemna, 2021). However,

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

while a superstructure may guarantee that the ancestral constraints are satisfied, it may also impose new constraints to unrelated nodes: to enforce an ancestral constraint, the superstructure may contain no set of edges that could lead to violating this constraint, even if there were also valid structures that used some of those edges.

Inspired by the recent advancements on structure learning where the search space is constrained by a parameter to make the problem tractable (Ordyniak and Szeider, 2013; Korhonen and Parviainen, 2015; Grüttemeier and Komusiewicz, 2022), we too approach the issue from the viewpoint of parameterized algorithms. In particular, we consider the *path cover number* of the ACG, which is the minimum number of possibly overlapping directed paths covering all nodes. Crucially, the path cover number of the ACG yields a bound for the number of *ideals* in it. An approach based on dynamic programming has been proven to solve the problem such that its complexity depends on the number of ideals of precedence constraints (Steiner, 1990; Parviainen and Koivisto, 2013). Therefore, we study whether learning becomes tractable over other classes of ancestral constraint graphs with polynomially many ideals.

Note that this parameterization differs from the others by being a property of the ACG and not the family of DAGs we can output, enabling more expressive structures. For instance, we can have arbitrarily large treewidth (complete graph) or long paths, which would not be possible for DAGs with, say, bounded treewidth or a bounded vertex cover number.

Under this parameterization, we also look into the related problems of weighted counting and sampling of networks structures proportionally to their scores. While these questions have been studied in the general case motivated by the Bayesian approach to learning Bayesian networks (Kuipers and Moffa, 2015, 2017; Talvitie et al., 2019), they have started gaining attention in the parameterized setting only recently (Harviainen and Koivisto, 2023).

In this paper, we give algorithms for the three problems of optimization, sampling, and counting that run in a polynomial time in the number of nodes for a PCG with a bounded path cover number. Positive ancestral constraints are non-decomposable (Chen et al., 2016; Li and van Beek, 2018) and the problems appear harder in this setting. With them, we require an additional assumption that the edges of the PACG are coverable with a bounded number of directed paths to obtain an efficient algorithm. For negative ancestral constraints, we show that the structure learning problem becomes NP-hard when the path cover number of the NACG is at least 2. Surprisingly, precedence constraints seem to be the easiest to deal with despite them being in the middle of the hierarchy of ancestral constraints. We summarize these results in Table 1.

Finally, we study special cases of the problems where we are able to achieve *fixed-parameter tractability*, that is, the problem can be solved in polynomial time for any bounded value of the parameter, with the exponent of the polynomial not depending on its value. First, we consider a setting where the PCG has a bounded path cover number and we have k additional nodes not included in the PCG. In this case, we show fixed-parameter tractability in k for all three problems. We prove similar results for precedence constraints and positive ancestral constraints for a special case of series-composable partial orders (Steiner, 1990) called bucket orders, which have been studied in a related setting by Koivisto and Parviainen (2010). In bucket orders, the nodes are partitioned into buckets of size bounded by a parameter b, and the nodes of a bucket are descendants of the nodes of all previous buckets in the ACG. Despite the path cover number being the size of the largest bucket, the number of ideals is only exponential in b, resulting in algorithms of running times that are polynomial in the size of the input and exponential in b. We conclude by discussing some potential directions for future work on incorporating structural information.

2 PRELIMINARIES

We start by recalling the basic definitions and algorithms for learning the structure of a Bayesian network. Then, we formally define the constraints used throughout the paper.

2.1 Bayesian Networks

Let $G = (N, A)$ be a DAG over n nodes where N is the node set and A is the edge set. The parent set of a node v is denoted by A_v. For each node $v \in N$, the input for the problems describes a *local score function* $s_v \colon 2^N \to \mathbb{R}_{\geq 0}$ that gives a score to its parents A_v. See, for example, the textbook of Koller and Friedman (2009) for a comprehensive summary of commonly used scores. The score of G is defined as the product over the local scores,

$$s(G) \coloneqq \prod_{v \in N} s_v(A_v).$$

Note that it is common to instead optimize the sum of logarithms of the scores, but using the product form is more appropriate for the other discussed problems. When the score combines a prior distribution and a likelihood function, it reflects how well a Bayesian network with that structure fits the data and the assumptions,

Table 1: Summary of the results. The path cover number of an ACG D is denoted by $\mathrm{pc}(D)$ and the line graph (Sec. 3.2) of D by D^*. Note that NP-hardness is a lower bound while the other results are upper bounds. The cases marked with "?" are not studied in this paper.

Constraint	DAG optimization	DAG counting	DAG sampling
Positive ancestral	$n^{O(\mathrm{pc}(D)\,\mathrm{pc}(D^*))}$ (Thm. 7)	$n^{O(\mathrm{pc}(D)\,\mathrm{pc}(D^*))}$ (Thm. 7)	$n^{O(\mathrm{pc}(D)\,\mathrm{pc}(D^*))}$ (Thm. 7)
Precedence	$n^{\mathrm{pc}(D)+O(1)}$ (Thm. 2)	$n^{3\,\mathrm{pc}(D)+O(1)}$ (Thm. 4)	$n^{3\,\mathrm{pc}(D)+O(1)}$ (Thm. 5)
Negative ancestral	NP-hard (Thm. 8)	?	?

motivating the problem of finding the maximum a posteriori structure:

DAG optimization (with constraints $\mathcal{C}$)
Output: The DAG $\arg\max_{G\in\mathcal{C}} s(G)$,

where—with abuse of notation—$G \in \mathcal{C}$ if G *satisfies* the constraints $\mathcal{C}$. For the general structure learning problem, the only constraint is that G has to be acyclic.

We make two assumptions in this paper that are common in the literature. Firstly, we assume the input uses non-zero encoding, where s_v assigns a score 0 to parents sets whose scores are not specified in the input. Further, we assume that the number of non-zero scores is polynomial in n[1]. For example, this constraint is satisfied by only computing local scores for parent sets of a bounded size. Nevertheless, the problem remains NP-hard even with parent sets of size at most two (Chickering, 1995).

Uncertainty plays an inherent part in machine learning, and thus having only the highest scoring DAG might not be enough. When taking the Bayesian approach to structure learning (Heckerman et al., 1995; Madigan and York, 1995), we perform our computations over multiple or possibly all structures, leading to the problems of sampling and weighted counting of DAGs.

DAG sampling (with constraints $\mathcal{C}$)
Output: A DAG $G \in \mathcal{C}$ with $\Pr(G) \propto s(G)$.

DAG counting (with constraints $\mathcal{C}$)
Output: The sum $\sum_{G\in\mathcal{C}} s(G)$.

These enable, for example, the computation of exact posterior probabilities of structural features and inference with model averaging. Analogously to optimization, counting DAGs is #P-hard in general (Harviainen and Koivisto, 2023).

[1]There are differing practices on whether the number of non-zero scores is incorporated in the time complexities (Parviainen and Koivisto, 2013; Grüttemeier and Komusiewicz, 2022) or assumed to be bounded by a polynomial (Harviainen and Koivisto, 2023). Here, we took the latter approach since it simplifies the presentation of the results and is how structure learning is often performed in practice, i.e., the local scores are computed for parent sets up to some fixed small size.

2.2 Optimization by Utilizing Ideals

The classic algorithm (Ott et al., 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006) for finding an optimal network structure relies on the idea that each DAG has a sink—a node with no outgoing edges. To find the optimal structure on the node set N, we search for the best structure on $N \setminus \{v\}$ for all v and find the best parent set for v from $N \setminus \{v\}$. Applying the algorithm recursively and memoizing the results leads to the time complexity of $2^n n^{O(1)}$. For the purposes of this paper, however, it will be more practical to approach the problem by using partial orders and ideals.

Recall that a DAG encodes a partial order $P \subseteq N \times N$ of the nodes: for each pair of nodes $v, w \in N$, let $vw \in P$ if and only if v is an ancestor of w. In a linear order L, any two nodes are comparable, that is, $vw \in L$ or $wv \in L$. A linear order is a linear extension of a partial order if $P \subseteq L$. We denote the set of linear extensions of P by $\mathcal{L}(P)$.

Suppose that we look for an optimal structure that is compatible with a given partial order, that is, the partial order and the DAG share a linear extension. Intuitively, if we know that a node v must precede another node u, then we should not search for optimal structures over subsets that contain u but not v. An ideal I of a partial order P of nodes is a subset of the nodes such that if $w \in I$ and $vw \in P$, then $v \in I$. The set of ideals of a partial order P is denoted by $\mathcal{I}(P)$. For an ideal $I \in \mathcal{I}(P)$, we let $\mathrm{opt}(I)$ be the largest product of local scores for nodes in I such that the parent sets are compatible with the partial order, that is,

$$\mathrm{opt}(I) \coloneqq \max_{L\in\mathcal{L}(P)} \prod_{v\in I} \max_{A_v\subseteq L_v} s_v(A_v),$$

where $L_v \coloneqq \{w \colon wv \in L\}$. Then, $\mathrm{opt}(I)$ obeys the following recurrence (Steiner, 1990):

$$\mathrm{opt}(I) = \max_{\substack{v\in I \\ I\setminus\{v\}\in\mathcal{I}(P)}} \mathrm{opt}(I \setminus \{v\}) \cdot \max_{A_v\subseteq I\setminus\{v\}} s_v(A_v).$$

Notably, $\mathrm{opt}(N)$ corresponds to an optimal network structure that shares a linear extension with P. The algorithm needs roughly $|\mathcal{I}(P)|$ time to compute $\mathrm{opt}(N)$

and reduces to the classical dynamic programming algorithm when $P = \emptyset$, since any subset of the nodes is an ideal.

2.3 Ancestral Constraints

In this paper, we consider three types of ancestral constraints over ordered pairs of nodes (v, w), where $v, w \in N$, that correspond to incorporating structural information obtained from, for example, domain knowledge:

Precedence: the node v must precede w in a topological order;

Positive: the node v must be an ancestor of w;

Negative: the node w cannot be an ancestor of v.

For the precedence constraints, there must be a topological order for which all the constraints hold simultaneously. Given a set of constraints of the same type, we gather them into a *precedence constraint graph* (PCG) or a *negative/positive ancestral constraint graph* (NACG/PACG), denoted by $D = (N, S)$, where a constraint (v, w) is represented as an edge from v to w.

The differences of these ancestral constraint graphs (ACGs) are illustrated in Figure 1. The graph in Figure 1(a) to be the constraint graph D. We see that the DAG (d) satisfies NACG D as (d) does not have a path from v_3 to v_1 or a path from v_4 to v_2. However, (d) does not satisfy PCG D because none of the topological orders of (d) has v_1 before v_3 and v_2 before v_4. Furthermore, (d) does not satisfy PACG D either because (d) does not have a path from v_1 to v_3 and a path from v_2 to v_4. The DAG (b) satisfies NACG D and PCG D but not PACG D because there is no path from v_1 to v_3 in (b). Finally, the DAG (c) satisfies NACG, PCG, and PACG D.

Hereinafter, we let our set of constraints $\mathcal{C}$ include the constraints imposed by the given ACG. We will specify the type of the ACG when it is not clear from context. An algorithm for optimization with a PCG was shown in Section 2.2.

We will quickly give alternative formulations for PCGs and PACGs. For PCGs, an equivalent definition would be that some linear extension of G is a linear extension of D or that the graph $(N, S \cup A)$ is acyclic. Alternative formulations for PACGs are having any linear extension of G be a linear extension of D or having D be a subgraph of the transitive closure of G. As an example of positive ancestral constraints, consider a case where v is observed to cause w directly or indirectly. Then, D could contain an edge from v to w, and G should have a directed path from v to w (e.g., $v \to u \to w$, suggesting that v causes u that causes w).

3 SMALL PATH COVERS

As observed before, the number of ideals of an ancestral constraint graph can be exponentially large in the number of nodes. However, structural properties of the ACG can be used to bound its number of ideals. Here, we consider the number of paths it takes to cover the nodes of the ancestral constraint graph, and present the time complexities in terms of that *parameter* and the number of nodes.

A *path cover* of a DAG is a set of directed paths such that each node is in at least one of the paths. The path cover number $\mathrm{pc}(D)$ of a DAG D is the size of its smallest path cover and is computable in polynomial time if it is not already given as input (Dilworth, 1950; Fulkerson, 1956; Cáceres et al., 2022). The path cover number serves two purposes for us. First, any set of nodes of size greater than $\mathrm{pc}(D)$ has at least two comparable nodes. A set without any comparable nodes is called an *antichain*, and there is a bijection between them and the ideals. The second useful property of $\mathrm{pc}(D)$ is that it bounds the number of ideals of the ancestral constraint graph:

Lemma 1 (Steiner (1990))**.** *The number of ideals a DAG D can have is at most $\big(1 + n/\,\mathrm{pc}(D)\big)^{\mathrm{pc}(D)}$.*

For instance, we get bounds $1 + n$ and 2^n at the two extremes where D is either a complete DAG or an empty DAG, respectively. In general, the bound is tight for DAGs with k chains of length n/k.

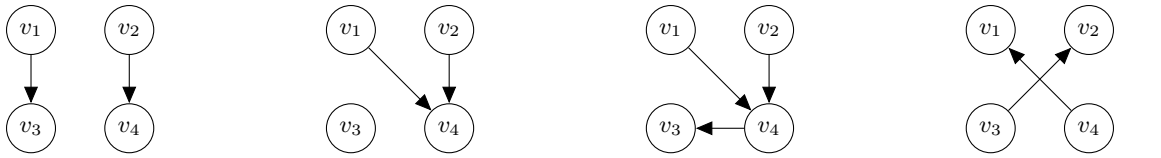
We emphasize that a small path cover number of the PCG does not mean that the possible DAGs are not expressive. Every PCG is satisfied by an empty DAG, some chain graph, and a complete DAG, demonstrating the wide variety of structures and the properties they may have.

3.1 Precedence Constraints

We start by observing a corollary of Lemma 1 and the algorithm of Section 2.2:

Theorem 2. *The* DAG OPTIMIZATION *problem with a PCG D can be solved in time $n^{\mathrm{pc}(D)+O(1)}$.*

Let us next turn our attention to DAG COUNTING with a PCG—an algorithm for sampling follows from that. In the general case without structural information, the state-of-the-art approach (Talvitie et al., 2019) for weighted counting is dynamic programming over partitions of nodes called *root-layerings* (Kuipers and Moffa, 2015, 2017). The root-layering $R = (R_1, R_2, \ldots, R_\ell)$ of a DAG is obtained by repeatedly removing the source nodes of the DAG: R_1 contains the source nodes, R_2 the source nodes of the graph induced by $N \setminus R_1$,



(a) An ACG D (b) Satisfies PCG D (c) Satisfies PACG D (d) Satisfies NACG D

Figure 1: An ACG (a) and three DAGs (b)–(d) that satisfy it depending on its type. If the ACG is a PCG, then the DAG in (b) and (c) satisfy the constraints. Similarly, the NACG is satisfied by (b)–(d) and the PACG by (c).

and so on. Then, we recursively solve subproblems where we ask for the total score of DAGs over nodes $R_{1:k} \coloneqq R_1 \cup R_2 \cup \cdots \cup R_k$ whose last layer is R_k.

Ideally, each layer of G would be narrow such that we had $|R_i| \leq f(\mathrm{pc}(D))$ for some f, since then there are at most $n^{f(\mathrm{pc}(D))}$ possible sets to consider for each layer. However, an empty graph G always satisfies the PCG, and for it $|R_1| = n$. Therefore, instead of computing the total score using the root-layerings of the DAGs $G = (N, A)$, we add the set of structural edges S from D to G to obtain $G^D \coloneqq (N, S \cup A)$ whose root-layering we consider. The layers of G^D are narrow as desired:

Lemma 3. *Each layer of the root-layering of G^D has size at most* $\mathrm{pc}(D)$.

Proof. No node on a layer can be an ancestor of another on the same layer by the definition of the root-layering. Thus, they are incomparable. The proof is completed by that $\mathrm{pc}(D) \geq \mathrm{pc}(G^D)$ and any subset of nodes of size greater than $\mathrm{pc}(G^D)$ must have a pair of comparable nodes in G^D. □

We say that a root-layering $R = (R_1, R_2, \ldots, R_\ell)$ satisfies the PCG if for every R_k with $1 \leq k \leq \ell$ and $v \in R_k$ we have that $S_v \subseteq R_{1:k-1}$, enforcing that the parents of v in the PCG appear on previous layers. Denote the set of root-layerings that satisfy the PCG by $\mathcal{R}$. If G satisfies the precedence constraints, the uniquely defined root-layering of G^D satisfies the PCG. We denote this layering by $r\big(G^D\big)$. Now, the total score of feasible DAGs can rewritten as a sum over root-layerings $R = (R_1, R_2, \ldots, R_\ell) \in \mathcal{R}$,

$$\sum_{G \in \mathcal{C}} s(G) = \sum_{R \in \mathcal{R}} \sum_{\substack{G \in \mathcal{C} \\ r\left(G^D\right) = R}} s(G)$$

$$= \sum_{R \in \mathcal{R}} \sum_{\substack{G \in \mathcal{C} \\ r\left(G^D\right) = R}} \prod_{k=1}^{\ell} \prod_{v \in R_k} s_v(A_v).$$

For R to be the root-layering of G^D, a node v that is on a layer R_{k+1} must have a parent on the layer R_k or the set $S_v \cap R_k$ cannot be empty; otherwise it would have been a source node in G^D after removing the nodes $R_{1:i}$ for some $i < k$. Layer R_1 is treated as a special case, since nodes there do not have parents. The total score of feasible DAGs can thus be further rewritten as

$$\sum_{R \in \mathcal{R}} \left(\prod_{v \in R_1} s_v(\emptyset) \right) \prod_{k=1}^{\ell-1} \prod_{v \in R_{k+1}} \sum_{\substack{A_v \subseteq R_{1:k} \\ (S_v \cup A_v) \cap R_k \neq \emptyset}} s_v(A_v).$$

To simplify the expression, define $\hat{s}_v(\emptyset, \emptyset) \coloneqq s_v(\emptyset)$ if $S_v = \emptyset$,

$$\hat{s}_v(R_{1:k}, R_k) \coloneqq \sum_{\substack{A_v \subseteq R_{1:k} \\ (S_v \cup A_v) \cap R_k \neq \emptyset}} s_v(A_v) \tag{1}$$

if $S_v \subseteq R_{1:k}$, and otherwise $\hat{s}_v(R_{1:k}, R_k) \coloneqq 0$. The total score then reduces to

$$\sum_{R \in \mathcal{R}} \prod_{k=0}^{\ell-1} \prod_{v \in R_{k+1}} \hat{s}(R_{1:k}, R_k).$$

Algorithmically, we can evaluate this sum by utilizing the following recurrence:

$$\mathrm{tot}(R_{1:k}, R_{k+1}) = \sum_{\substack{R_k \subseteq R_{1:k} \\ |R_k| \leq \mathrm{pc}(D)}} \mathrm{tot}(R_{1:k} \setminus R_k, R_k) \prod_{v \in R_{k+1}} \hat{s}_v(R_{1:k}, R_k)$$

with $\mathrm{tot}(\emptyset, \emptyset) = 1$. By induction, it is straightforward to see that $\mathrm{tot}(R_{1:k}, R_{k+1})$ is the total score of different combinations of parent sets on $R_{1:k+1}$ whose last layer is R_{k+1} and which are compatible with the partial order. Thus, $\mathrm{tot}(N, \emptyset)$ equals the total score of feasible DAGs.

Finally, observe that $R_{1:k+1}$ must be an ideal of D for $\mathrm{tot}(R_{1:k}, R_{k+1})$ to be positive: Assume that there are nodes $v \in N \setminus R_{1:k+1}$ and $w \in R_{1:k+1}$ such that v is an ancestor of w in D. Further assume—without loss of generality—that v is a direct parent of w. Then, by definition, $\hat{s}_w(\cdot, \cdot) = 0$, which implies that $\mathrm{tot}(R_{1:k}, R_{k+1}) = 0$. Therefore, if $R_{1:k+1}$ is not an ideal of D, we can immediately return 0 for $\mathrm{tot}(R_{1:k}, R_{k+1})$ without evaluating the recurrence further.

Lemma 1 implies that D has at most $n^{\mathrm{pc}(D)+O(1)}$ ideals, and so there are at most that many values $R_{1:k+1}$ to consider. Further, R_{k+1} and R_k have size at most $|\,\mathrm{pc}(D)|$, meaning each of the $n^{2\,\mathrm{pc}(D)+O(1)}$ values of $\mathrm{tot}(R_{1:k}, R_{k+1})$ takes $n^{\mathrm{pc}(D)+O(1)}$ time to compute. Hence, we get the following theorem:

Theorem 4. *The* DAG COUNTING *problem with a PCG D can be solved in time $n^{3\,\mathrm{pc}(D)+O(1)}$.*

A weighted counting algorithm gives us a sampling algorithm straightforwardly because of the self-reducible nature of the problem (Talvitie et al., 2019). We start by sampling R_ℓ proportionally to $\mathrm{tot}(N \setminus R_\ell, R_\ell)$. Then, the kth layer for $k = \ell - 1, \ell - 2, \ldots, 1$ is sampled with probabilities proportional to

$$\mathrm{tot}(R_{1:k} \setminus R_k, R_k) \prod_{v \in R_{k+1}} \hat{s}_v(R_{1:k}, R_k). \tag{2}$$

Afterwards, the parents of each node v are sampled proportionally to $s_v(A_v)$ such that they satisfy the constraints $A_v \subseteq R_{1:k}$ and $(S_v \cup A_v) \cap R_k \neq \emptyset$. By precomputing the $n^{3\,\mathrm{pc}(D)+O(1)}$ unnormalized probability masses in Equation (2), one can apply binary search to draw samples in time $n^{O(1)}$.

Theorem 5. *The* DAG SAMPLING *problem with a PCG D can be solved in preprocessing time $n^{3\,\mathrm{pc}(D)+O(1)}$ and sampling time $n^{O(1)}$.*

Additionally, observe that replacing the summation by maximization in Equation (1) gives an alternative (albeit slower) algorithm for the DAG OPTIMIZATION. We will use this fact to give an algorithm for all three problems with PACG simultaneously.

3.2 Positive Ancestral Constraints

We proceed directly to giving an algorithm for counting, since adapting it to sampling and optimization is straightforward. Again, our approach is based on root-layerings. However, this time we need to ensure that the DAGs contain the necessary directed paths implied by the PACG. Without loss of generality, assume all edges that would follow from transitivity are removed from the PACG as this simplifies the proofs. This does not change the path cover number of the PACG.

We start with a couple of observations. First, for a DAG G to satisfy a PACG D, it suffices to ensure that, for every node v, all of its parents in D are its ancestors in G: all other ancestors of v in D must already be connected to one such node. Since any two such parents are incomparable, there can be at most $\mathrm{pc}(D)$ of them.

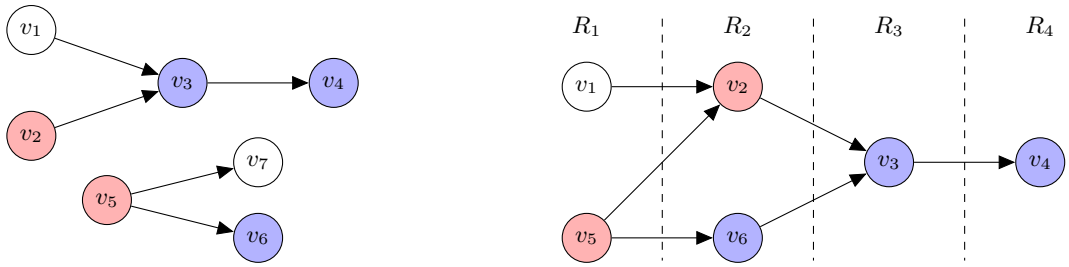
Suppose that the first k layers of the root-layering of G satisfying D correspond to a partial root-layering $R_{1:k}$. The second observation is that the union of the set $\{w\}$ and the descendants of w in $R_{1:k}$ denoted by M_w can be uniquely identified by a subset of nodes of size at most $\mathrm{pc}(D)$. Take the subgraph of the PACG induced by M_w and initialize the *head* H_w of w by M_w. Then, while H_w contains two nodes such that one is an ancestor of the other in D, remove the descendant from H_w. Since the remaining nodes in the sets are incomparable, there can be at most $\mathrm{pc}(D)$ of them.

It remains to show that M_w can be uniquely reconstructed from H_w given the partial root-layering $R_{1:k}$ and the PACG D. By definition, H_w is a subset of M_w, and any node $u \in M_w \setminus H_w$ must have an ancestor from H_w in D. On the other hand, any node in $R_{1:k}$ with an ancestor from H_w in D is a descendant of w by the definition of $M_w \supseteq H_w$. These concepts are illustrated in Figure 2. Note that the sets M_w and H_w are functions of the DAG G, its partial root-layering $R_{1:k}$, and the PACG even though we omit them from the notation for the sake of readability.

We are now ready to start describing our algorithm. Like before, we have $R_{1:k}$ and R_{k+1} as the arguments of the dynamic programming recurrence, but we also need some additional arguments to keep track on which paths have to be present. More precisely, we include H_w if $w \in R_{1:k}$ has a child in D that is not in $R_{1:k}$. We denote the set of such nodes w by W and define $\mathcal{H} = \{H_w \mid w \in W\}$.

Recall that $R_{1:k}$ must be an ideal of D—this allows us to ignore partial root-layerings that contribute nothing to the total weight in the dynamic programming. There are at most $\mathrm{pc}(D)$ source nodes in the subgraph of D induced by $N \setminus R_{1:k}$, and R_{k+1} has to be a subset of them. Problematically, the size of W can be linear in n. To see this, consider a graph with $2n$ nodes of $v_{i,c}$ with $i \in \{1, 2, \ldots, n\}$ and $c \in \{1, 2\}$ such that the PACG has edges $v_{i,1} \to v_{i,2}$ and $v_{i,c} \to v_{i+1,c}$ for all i and c. Then, for a DAG $v_{1,1} \to v_{2,1} \to \ldots \to v_{n,1} \to v_{1,2} \to v_{2,2} \to \ldots \to v_{n,2}$ and partial root layering $\{v_{i,1} \mid i \in \{1, 2, \ldots, n\}\}$, every node $v_{i,1}$ needs to be included in W. Despite this, we first give a general algorithm for any PACG, and only then proceed to give a workaround by considering another parameterization.

Due to how involved the recurrence is, we omit presenting it formally, and instead describe it verbally and as pseudocode in Algorithm 1 of supplementary materials. Suppose you have computed the total weight of partial root-layerings on $R_{1:k}$ with the last layer being R_k such that the relevant head sets are $\mathcal{H}$. Next, consider some set of nodes R_{k+1} for the next layer, and recall that any such node v needs to have a parent from R_k in addition to a node from M_w for every $w \in S_v$. We iterate over all valid parent set assignments for nodes of R_{k+1} simultaneously, and then compute the sets $\mathcal{H}'$:



(a) An ancestral constraint graph (b) A partial root-layering

Figure 2: An ancestral constraint graph and a partial root-layering of a DAG satisfying it. For the PACG and this DAG, we have $H_{v_5} = \{v_2, v_5\}$ (red) and $M_{v_5} = \{v_2, v_3, v_4, v_5, v_6\}$ (blue). To ensure a path from v_5 to v_7, it is sufficient and necessary for v_7 to have at least one parent from M_{v_5} when it gets added to the final layer R_5.

If $v \in R_{k+1}$ is a parent of some node in D, then add $H_v = \{v\}$ to $\mathcal{H}'$. If w with $H_w \in \mathcal{H}$ is a parent of some node in $N \setminus R_{1:k+1}$, insert H_w to $\mathcal{H}'$. Finally, if $v \in R_{k+1}$ has a parent in $H_w \in \mathcal{H}'$ but no ancestors from D in H_w, add v to $H_w \in \mathcal{H}'$. It remains to add the total weight of partial root-layerings on $R_{1:k}$ multiplied by the scores of the R_{k+1} to the dynamic programming value indexed by the set of used nodes $R_{1:k+1}$, the latest layer R_{k+1}, and the new set of heads $\mathcal{H}'$. In the end, a the dynamic programming states with $R_{1:\ell} = N$ and $W = \mathcal{H} = \emptyset$ corresponds to the total weight of all DAGs satisfying the constraints.

The proof of correctness is similar as with PCGs but more involved. Essentially, each DAG G satisfying the PACG corresponds to a unique root-layering and sequences of sets W and $\mathcal{H}$, and the recurrence ensures all constraints are satisfied. After the computation of the dynamic programming states, stochastic backtracking enables sampling, and replacing addition by maximization lets us search for the optimal structure.

However, as mentioned before, the size of W can be linear in n for some instances, and thus the running time would become super-exponential. Therefore, we consider an alternative parameterization.

The directed line graph D^* of D is a directed graph obtained by transforming each edge of D into a node and connecting two such nodes if they correspond to edges v_1v_2 and v_3v_4 with $v_2 = v_3$. Two edges are comparable in the line graph if there is a directed path containing both edges in D.

Lemma 6. *We have* $|W| \leq \mathrm{pc}(D^*)$ *for any partial root-layering* $R_{1:k}$ *of non-zero weight.*

Proof. Any two edges in D with the parent in $R_{1:k}$ and the child in $N \setminus R_{1:k}$ must be incomparable in D^*. Additionally, any set of incomparable edges must have size at most $\mathrm{pc}(D^*)$. As $w \in W$ only if $w \in R_{1:k}$ and it has a child in $N \setminus R_{1:k}$, there can be at most $\mathrm{pc}(D^*)$ such nodes w. □

Thus, we only need to consider sets W of size at most $\mathrm{pc}(D^*)$, and describing each H_w requires at most $\mathrm{pc}(D)$ nodes. This leads to following complexities:

Theorem 7. *The* DAG optimization, DAG sampling, *and* DAG counting *problems with a PACG* D *can be solved in (preprocessing and sampling) time* $n^{O\left(\mathrm{pc}(D)\,\mathrm{pc}(D^*)\right)}$.

Here, we let the path cover number of an empty graph be 1 for notational convenience, which happens only if D has no edges. Note that the relationship between $\mathrm{pc}(D)$ and $\mathrm{pc}(D^*)$ is somewhat unclear. If the PACG has no isolated nodes, then showing that $\mathrm{pc}(D) \leq \mathrm{pc}(D^*)$ is straightforward. However, the earlier example shows a graph with $\mathrm{pc}(D) = 2$ and $\mathrm{pc}(D^*) = n/2$. Still, it would seem reasonable to assume that narrow PACGs typically have narrow line graphs.

3.3 Negative Ancestral Constraints

With negative constraints, optimization is hard even for simple instances (proof deferred to Supplement):

Theorem 8. *The* DAG optimization *problem with a NACG of path cover number at least* 2 *is NP-hard.*

Since optimization is already hard, we do not study sampling and counting with a NACG in this paper.

4 FIXED-PARAMETER TRACTABILITY

If the problems were *fixed-parameter tractable* when parameterized by $\mathrm{pc}(D)$ and $\mathrm{pc}(D^*)$ then the exponent of n in the time complexities would not depend on them. We conjecture the problems to be W[1]-hard under these parameters, making fixed-parameter tractability unlikely (Downey and Fellows, 1995). Thus, we explore several alternative parameters with which we achieve fixed-parameter tractability.

4.1 Additional Constrained Nodes

First, assume that we focus only on instances where $\mathrm{pc}(D) = O(1)$, meaning that all the algorithms for PCGs run in polynomial time for all $D = (N, S)$. Then, suppose we want to add k additional nodes K to the DAG such that constraints concerning them do not depend on the nodes N. In other words, we are given D as well as another PCG $D' = (K, S')$ with $S' \subseteq K \times K$, and we wish to find a DAG G on nodes $N \cup K$ satisfying a PCG $(N \cup K, S \cup S')$ referred to as the k-PCG.

A k-PCG has $|\mathcal{I}(D')| \cdot |\mathcal{I}(D)| \leq 2^k n^{O(1)}$ ideals. This bound leads to fixed-parameter tractability of the problems, because all presented time complexities for PCGs are polynomial in the number of ideals. More formally, we get the following corollary:

Corollary 9. *The* DAG OPTIMIZATION*,* DAG SAMPLING*, and* DAG COUNTING *problems with a k-PCG can be solved in time $(2^k n)^{O(1)}$.*

The problem appears harder for positive constraints as the PACG for the k nodes could have two interlinked chains, forcing us to perform computations over sets W of size linear in k.

4.2 Bucket Orders

An alternate direction where we achieve fixed-parameter tractability does not require assuming a bounded path cover number. Instead, we further restrict the structure of D: We consider *bucket orders*, where the nodes are partitioned into buckets $B_1, B_2, \ldots, B_\ell$ of size at most b with constraints on how the nodes in different buckets can be connected (Steiner, 1990). We represent this as an ancestral constraint graph with the parent set of every node in B_{i+1} being B_i. Note that these *bucket ancestral constraint graphs* have a path cover number at most b, but their number of ideals is at most $2^b n$. In other words, $\mathrm{pc}(D)$ can be misleading estimator for the running time.

Importantly, we can run our computations over these buckets independently of each other for PCGs, and with PACGs we care only about adjacent buckets.

For optimization with a bucket PCG, we run the classical exponential-time algorithm (Ott et al., 2004; Singh and Moore, 2005; Silander and Myllymäki, 2006) for each bucket with the only difference being that the nodes may have parents also from previous buckets. The same applies to counting and sampling (Talvitie et al., 2019). Thus, we have the following results:

Corollary 10. *The* DAG OPTIMIZATION *problem with a bucket PCG can be solved in time $2^b n^{O(1)}$.*

Corollary 11. *The* DAG SAMPLING *problem with a bucket PCG can be solved in preprocessing time $3^b n^{O(1)}$ and sampling time $2^b n^{O(1)}$.*

Corollary 12. *The* DAG COUNTING *problem with a bucket PCG can be solved in time $3^b n^{O(1)}$.*

The case with a bucket PACG is slightly more complicated as we also need to ensure that the nodes in B_i are ancestors of nodes in B_{i+1}. This is achieved with the help of the following observation:

Lemma 13. *Let $B_1, B_2, \ldots, B_\ell$ be a bucket order and G a DAG where the parents of every $v \in B_i$ are from $B_{1:i}$. Then, G satisfies the bucket PACG if and only if the sinks of the subgraph induced by B_i are parents of the source nodes of the subgraph induced by B_{i+1} for all i.*

Proof. Assume G satisfies the bucket PACG. Then, any sink of the subgraph induced by B_i is a parent of the source nodes of the subgraph induced by B_{i+1}, since otherwise it cannot be their ancestor.

Assume next that the sinks of the subgraph induced by B_i are parents of the source nodes of the subgraph induced by B_{i+1}. Then, for $v \in B_i$ and $w \in B_{i+1}$, there is a sink in B_i and a source node in B_{i+1} such that v is an ancestor of the sink and w is a descendant of the source. Further, as any node in any B_i is an ancestor of all nodes in B_{i+1}, they are ancestors of nodes in B_j with $j > i$. □

We will proceed as follows: First, compute the optimal DAG or the total weight of bucket B_1 such that its sinks are S_1. Then, we do the same for B_2 and ensure its source nodes are connected at least to S_1, and so on. With minor modifications to the dynamic programming, we solve the problems in time $3^b n^{O(1)}$.

Corollary 14. *The* DAG OPTIMIZATION *problem with a bucket PACG can be solved in time $3^b n^{O(1)}$.*

Corollary 15. *The* DAG SAMPLING *problem with a bucket PACG can be solved in preprocessing time $3^b n^{O(1)}$ and sampling time $2^b n^{O(1)}$.*

Corollary 16. *The* DAG COUNTING *problem with a bucket PACG can be solved in time $3^b n^{O(1)}$.*

Note that we could add arbitrary constraints (i.e., not only ancestral) between the nodes within each bucket, and the problems would remain fixed-parameter tractable. Even though the number of DAGs is super-exponential in the number of nodes, the size of each bucket is bounded by b, and thus we could iterate over all possible DAGs for each bucket. Then, the states of our dynamic programming would depend only on the DAGs of the current bucket and the previous bucket.

5 CONCLUDING REMARKS

We presented new parameterized results on learning, sampling, and weighted counting of Bayesian network structures that must incorporate a set of ancestral constraints. When the PCG has a bounded path cover number, all these algorithms run in a polynomial time. For positive constraints, we required the assumption that also the line graph had a small path cover. Additionally, the problems become fixed-parameter tractable when the ancestral constraint graph follows a bucket order of buckets of a bounded size b, or we add k nodes to a PCG with precedence constraints of their own. On the other hand, the optimization problem is NP-hard even with a NACG of path cover number 2.

By no means do we claim our bounds are tight, especially with positive ancestral constraints. Instead, our goal was to lay the groundwork for future studies by exploring this border between tractable and intractable, as it is unclear how expressive we can keep the class of valid DAGs while maintaining tractability. Common to all presented algorithms is their dependence on the number of ideals of the ancestral constraint graph. However, obtaining algorithms that are sublinear in their number of ideals seems difficult, and such an algorithm would yield a faster algorithm for unconstrained structure learning.

Other open questions relate on incorporating multiple types of constraints simultaneously. For example, we could be given both a PCG and a PACG as an input, and the graph would have to satisfy both of them. Is it sufficient for only one of the ancestral constraint graphs to have a bounded path cover number, or is the problem intractable even if both of them are bounded?

Acknowledgements

This research was partially supported by the Research Council of Finland, Grant 351156.

References

Cáceres, M., Cairo, M., Mumey, B., Rizzi, R., and Tomescu, A. I. (2022). Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 359–376. SIAM.

Chen, E. Y., Shen, Y., Choi, A., and Darwiche, A. (2016). Learning Bayesian networks with ancestral constraints. In *Advances in Neural Information Processing Systems 29, NeurIPS 2016*, pages 2325–2333.

Chickering, D. M. (1995). Learning Bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer.

de Campos, C. P. and Ji, Q. (2011). Efficient structure learning of Bayesian networks using constraints. *J. Mach. Learn. Res.*, 12:663–689.

Dilworth, R. (1950). A decomposition theorem for partially ordered sets. *Ann. Math.*, pages 161–166.

Downey, R. G. and Fellows, M. R. (1995). Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131.

Fellows, M. R., Hermelin, D., Rosamond, F. A., and Vialette, S. (2009). On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61.

Fulkerson, D. R. (1956). Note on Dilworth's decomposition theorem for partially ordered sets. *Proc. Am. Math. Soc.*, 7(4):701–702.

Ganian, R. and Korchemna, V. (2021). The complexity of Bayesian network learning: Revisiting the superstructure. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, pages 430–442.

Grüttemeier, N. and Komusiewicz, C. (2022). Learning Bayesian networks under sparsity constraints: A parameterized complexity analysis. *J. Artif. Intell. Res.*, 74:1225–1267.

Harviainen, J. and Koivisto, M. (2023). Revisiting Bayesian network learning with small vertex cover. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2023*.

He, C., Di, R., and Tan, X. (2023). Bayesian network structure learning using improved A* with constraints from potential optimal parent sets. *Mathematics*, 11(15).

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Mach. Learn.*, 20(3):197–243.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York.

Koivisto, M. and Parviainen, P. (2010). A space-time tradeoff for permutation problems. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 484–492. SIAM.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.

Korhonen, J. H. and Parviainen, P. (2015). Tractable Bayesian network structure learning with bounded

vertex cover number. In *Advances in Neural Information Processing Systems 28: NeurIPS 2015*, pages 622–630.

Kuipers, J. and Moffa, G. (2015). Uniform random generation of large acyclic digraphs. *Stat. Comput.*, 25(2):227–242.

Kuipers, J. and Moffa, G. (2017). Partition MCMC for inference on acyclic digraphs. *J. Am. Stat. Assoc.*, 112(517):282–299.

Li, A. and van Beek, P. (2018). Bayesian network structure learning with side constraints. In *Proceedings of the Ninth International Conference on Probabilistic Graphical Models, PGM 2018*, volume 72 of *Proceedings of Machine Learning Research*, pages 225–236. PMLR.

Madigan, D. and York, J. (1995). Bayesian graphical models for discrete data. *Int. Stat. Rev.*, 63:215–232.

Ordyniak, S. and Szeider, S. (2013). Parameterized complexity results for exact Bayesian network structure learning. *J. Artif. Intell. Res.*, 46:263–302.

Ott, S., Imoto, S., and Miyano, S. (2004). Finding optimal models for small gene networks. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 557–567.

Parviainen, P. and Koivisto, M. (2013). Finding optimal Bayesian networks using precedence constraints. *J. Mach. Learn. Res.*, 14(42):1387–1415.

Perrier, E., Imoto, S., and Miyano, S. (2008). Finding optimal Bayesian network given a super-structure. *J. Mach. Learn. Res.*, 9(74):2251–2286.

Peruvemba Ramaswamy, V. and Szeider, S. (2022). Learning large Bayesian networks with expert constraints. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022*, volume 180 of *Proceedings of Machine Learning Research*, pages 1592–1601. PMLR.

Silander, T. and Myllymäki, P. (2006). A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence, UAI 2006*. AUAI Press.

Singh, A. and Moore, A. (2005). Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University, School of Computer Science.

Steiner, G. (1990). On the complexity of dynamic programming for sequencing problems with precedence constraints. *Annals of Operations Research*, 26:103–123.

Talvitie, T., Vuoksenmaa, A., and Koivisto, M. (2019). Exact sampling of directed acyclic graphs from modular distributions. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 965–974. AUAI Press.

Yuan, C., Malone, B. M., and Wu, X. (2011). Learning optimal Bayesian networks using A* search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 2186–2191. AAAI Press.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] The time complexities of all algorithms are given up to polynomial factors. Space complexities match the time complexities up to polynomial factors.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes] The proofs are complete but are sometimes given before the theorem when the used techniques are described. In such cases, a separate proof environment afterwards is omitted to avoid repetition.
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Not Applicable]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

On Tractability of Learning Bayesian Networks with Ancestral Constraints: Supplementary Materials

A COUNTING WITH POSITIVE ANCESTRAL CONSTRAINTS

Algorithm 1 contains the pseudocode for computing the total weight of DAGs that satisfy the PACG $D = (N, S)$. Here, $\mathrm{tot}(R_{1:k}, R_k, W, \mathcal{H})$ corresponds to the total weight of partial root-layerings $R_{1:k}$ such that the latest layer is R_k and exactly the nodes $w \in W$ have children in D that are in $N \setminus R_{1:k}$ and have heads $H_w \in \mathcal{H}$. The total weight of all DAGs satisfying D can be written as

$$\sum_{R_\ell \in \mathcal{I}(D)} \mathrm{tot}(N, R_\ell, \emptyset, \emptyset).$$

We denote the set of antichains in D by $\mathcal{A}(D)$. Recall that it is sufficient to consider sets W of size at most the path cover number $\mathrm{pc}(D^*)$ of the line graph D^* of D.

Algorithm 1: DAG COUNTING with a PACG

Initialize $\mathrm{tot}(R_{1:k}, R_k, W, \mathcal{H}) \leftarrow 0$ for all $R_{1:k}, R_{k+1}, W, \mathcal{H}$;
for $R_{1:k} \in \mathcal{I}(D)$ **do**
  for $R_{k+1} \subseteq N \setminus R_{1:k}$ *with* $S_v \in R_{1:k}$ *for each* $v \in R_{k+1}$ **do**
    for $W \subseteq R_{1:k}$ **do**
      for $\mathcal{H} \in \bigtimes_{w \in W} \{H_w \subseteq R_{1:k} \mid w \in H_w \in \mathcal{A}(D)\}$ **do**
        for $A_v \subseteq R_{1:k}$ for all $v \in R_{k+1}$ **do**
          if $\exists v \in R_{k+1}, w \in S_v$ with $M_w \cap A_v = \emptyset$ **then**
            continue;
          Initialize $W' \leftarrow \emptyset$;
          Initialize $\mathcal{H}' \leftarrow \emptyset$;
          for $v \in R_{k+1}$ **do**
            if v has children in D **then**
              Insert v to W';
              Insert $H_v = \{v\}$ to $\mathcal{H}'$;
          for $w \in W$ **do**
            if $\exists u \in N \setminus R_{1:k+1}$ with $w \in S_u$ **then**
              Insert w to W';
              Insert $H_w \in \mathcal{H}$ to $\mathcal{H}'$;
              for $v \in R_{k+1}$ **do**
                if $M_w \cap A_v \neq \emptyset$ and no ancestor of v from D is in H_w **then**
                  Insert v to $H_w \in \mathcal{H}'$;
          $\mathrm{tot}(R_{1:k+1}, R_{k+1}, W', \mathcal{H}') \leftarrow$
            $\mathrm{tot}(R_{1:k+1}, R_{k+1}, W', \mathcal{H}') + \mathrm{tot}(R_{1:k}, R_k, W, \mathcal{H}) \cdot \prod_{v \in R_{k+1}} s_v(A_v)$;

B PROOF OF THEOREM 8

Proof. Recall that a set of vertices is a clique in an undirected graph if all its vertices are adjacent to each other. The problem of finding the largest clique of a graph is NP-hard (Karp, 1972). If we could efficiently decide whether a graph contains a clique of size k, then we could solve the Max Clique problem efficiently. On the other hand, the W[1]-hard k-Clique problem is reducible to the Multicolored k-Clique problem (Fellows et al., 2009), where in addition to an undirected graph $G' = (V, E)$, we are given a coloring of its vertices $c\colon V \to [k]$, and have to decide if the graph contains a k-clique whose vertices are of distinct colors. We show that a polynomial-time algorithm to the DAG optimization problem with a NACG of path cover number at least 2 would imply a polynomial-time algorithm to the Multicolored k-Clique problem, and thus to Max Clique.

Arbitrarily index the vertices of G' as $v_1, v_2, \ldots, v_n$ and the edges as $e_1, e_2, \ldots, e_m$, and represent them as nodes in DAG optimization. We refer to these as the *vertex nodes* and *edge nodes*. Add one node C_i for each color whose purpose is to choose a vertex of color i from a multicolored k-clique. For the other vertices, we add a node P that they should pick as a parent if they do not belong to the chosen k-clique.

Construct a NACG that is the transitive closure of the two chains $v_1 \leftarrow v_2 \leftarrow \cdots \leftarrow v_n$ and

$$P \leftarrow C_1 \leftarrow C_2 \leftarrow \cdots \leftarrow C_k \leftarrow e_1 \leftarrow e_2 \leftarrow \cdots \leftarrow e_m,$$

and assign the following local scores (and let others be zero):

$s_P(\emptyset) = 1$;

$s_{v_i}(\emptyset) = 1$;

$s_{v_i}(P) = 2^{m+1}$;

$s_{e_i}(\emptyset) = 1$;

$s_{e_i}(\{u, v\}) = 2$ if $e_i = \{u, v\}$;

$s_{C_i}(v) = 1$ if $c(v) = i$.

Assume G' contains a k-clique whose nodes are of distinct colors. Without loss of generality, assume those nodes to be $v_1, v_2, \ldots, v_k$. Then, the DAG G with parent sets

$G_P = \emptyset$;

$G_{v_i} = \emptyset$ if $i \leq k$;

$G_{v_i} = P$ if $i > k$;

$G_{e_i} = \{u, v\}$ if $e_i = \{u, v\}$ with $u, v \in \{v_1, v_2, \ldots, v_k\}$;

$G_{e_i} = \emptyset$ if $e_i = \{u, v\}$ with $u \notin \{v_1, v_2, \ldots, v_k\}$ or $v \notin \{v_1, v_2, \ldots, v_k\}$;

$G_{C_i} = v_i$

attains score $2^{(n-k)(m+1)+\binom{k}{2}}$. A DAG resulting from this construction is illustrated in Figure 3.

It remains to show that this score cannot be reached if G' does not contain a multicolored k-clique. Each C_i has to have a vertex node as a parent, and thus the product of the local scores of the vertex nodes is at most $2^{(n-k)(m+1)}$, for otherwise P is an ancestor of some C_i. Then, the factor $2^{\binom{k}{2}}$ would have to come from the nodes $e_1, e_2, \ldots, e_m$. The parents of e_i can only be nodes that are not children of P, and there are at most k of them. However, if $\binom{k}{2}$ nodes e_i have parent set from the parents of C_j, then the parents of C_j would form a k-clique. □

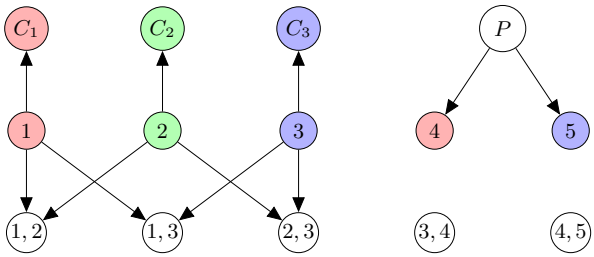


Figure 3: The optimal DAG in the reduction from Multicolored k-Clique to DAG optimization with a NACG if a multicolored k-clique exists in the original instance. Nodes with an integer represent the vertices of the original instance and pairs of integers correspond to its edges.