Revisiting Bayesian Network Learning with Small Vertex Cover

Abstract

The problem of structure learning in Bayesian networks asks for a directed acyclic graph (DAG) that maximizes a given scoring function. Since the problem is NP-hard, research effort has been put into discovering restricted classes of DAGs for which the search problem can be solved in polynomial time. Here, we initiate investigation of questions that have received less attention thus far: Are the known polynomial algorithms close to the best possible, or is there room for significant improvements? If the interest is in Bayesian learning, that is, in sampling or weighted counting of DAGs, can we obtain similar complexity results? Focusing on DAGs with bounded vertex cover number—a class studied in Korhonen and Parviainen's seminal work (NIPS 2015)—we answer the questions in the affirmative. We also give, apparently the first, proof that the counting problem is #P-hard in general. In addition, we show that under the vertex-cover constraint counting is #W[1]-hard.

1 INTRODUCTION

The structure of a Bayesian networks is a directed acyclic graph (DAG). The task of learning the DAG from given data is often formulated as maximization of some appropriate scoring function. Common scoring functions are decomposable, meaning that the score of a DAG is obtained as a sum (or product) of vertex-wise local scores, each of which only depends on the vertex and its parents in the graph. This structural property—which isolates the combinatorial problem from the specifics of the scoring function and the data—has motivated fruitful algorithmic research. The problem is known to be NP-hard [Chickering, 1995] even if every vertex is allowed to have at most two parents. On the other hand, an optimal n-vertex DAG can be found by dynamic programming over vertex subsets in time $O(2^n n^2)$ under any indegree or other local constraints [Ott et al., 2004, Singh and Moore, 2005, Silander and Myllymäki, 2006]. For almost two decades, we have seen essentially no progress in the worst-case time bound, albeit numerous algorithms have been developed for heuristic search.

Given that the problem is hard in general, we may ask whether it becomes tractable if we restrict the search space by some constraints (other than the plain maximum indegree). Taking the viewpoint of *parameterized complexity*, the question is whether the complexity of the problem can be controlled by some parameter of the input or output. While restricting the input never increases the complexity, restricting the output (i.e., the search space) may increase or decrease the complexity. For example, parameterizing by the *treewidth* of the DAG appears to only make the problem harder [Korhonen and Parviainen, 2013]. On the other hand, parameterizing by the *vertex cover number* renders the problem easier [Korhonen and Parviainen, 2015]. Other positive results are known for polytrees that are close to branchings [Gaspers et al., 2015] and for DAGs with bounded *feedback edge number* [Ganian and Korchemna, 2021]. Upper bounds can often be complemented with lower bounds, that is, parameterized hardness results within the *W-hierarchy* [Downey and Fellows, 1995]. Typically, the time complexities are polynomial in the number of vertices for any constant value of the parameter: Grüttemeier and Komusiewicz [2022] provide a summary of many complexity results for different parameters.

In this paper, we put forward two questions that have received little attention in previous works on structure learning in Bayesian networks. First, we ask whether the known parameterized algorithms are close to the best possible. For example, if an algorithm with running time $O(T)$ is known, can we find an algorithm that runs in time $O(\sqrt{T})$? Such a speedup would be a significant *quantitative* improvement, even if it did not affect the *qualitative* complexity classification, which has been the primary interest of previous

Submitted to the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023). **To be used for reviewing only**.

works. Second, we ask to what extent the parameterized complexity results for the optimization problem in question can be transferred to related problems of sampling and weighted counting of DAGs. These variants are motivated chiefly by the Bayesian approach to learning Bayesian networks [Heckerman et al., 1995, Madigan and York, 1995]. Both exact algorithms [Koivisto and Sood, 2004, Tian and He, 2009, Talvitie et al., 2019, Koivisto and Röyskö, 2020] and numerous sampling-based approximate methods (see, e.g., Friedman and Koller [2003], Niinimaki et al. [2016], Kuipers and Moffa [2017] and references therein) have been developed. However, these works have not exercised the parameterized complexity viewpoint.

To initiate the investigation of these question, we focus on parameterization by the vertex cover number, studied in the seminal work by Korhonen and Parviainen [2015]. They gave an algorithm running in time $n^{2k+O(1)}$ for any fixed vertex cover number k. Furthermore, they showed that the parameterized problem is W[1]-hard, suggesting that the obtained time bound is qualitatively perhaps the best possible: the exponent of n must depend on k.

After formalizing the setup in Section 2, we begin in Section 3 by giving an algorithm that finds an optimal DAG in time $n^{k+O(1)}$, thus achieving a nearly quadratic improvement. In Section 4, we turn to the counting variant, with several results: We start by showing that the basic, unconstrained problem is #P-hard; while this result may be unsurprising, to our knowledge, it has not been proven before. We then give an analogous #W[1]-hardness result for the parameterized variant, but also an algorithm running in time $n^{2k+O(1)}$. In Section 5, we consider the sampling variant. While the handling of duplicates appears to make counting more difficult than maximization, we observe that by having a control of the number of duplicates enables relatively efficient rejection sampling. Finally, we end the paper by pointing directions to future research in Section 6.

2 PROBLEM FORMULATIONS

A Bayesian network is a graphical representation of a multivariate probability distribution. Its structure is given as a DAG $G = (V, E)$, where each vertex in V corresponds to one random variable and the edge set E encodes conditional independence relations between the variables. If (u, v) is an edge in G, we call u a *parent* of v and v a *child* of u. We write G_v for the *parent set* of v in G, i.e., $G_v = \{u \in V \mid (u, v) \in E\}$. The joint distribution of the variables is obtained by specifying a univariate conditional probability distribution for each vertex v given its parent set G_v, and then taking the product of these distributions.

In score-based structure learning of Bayesian networks, every possible DAG G is associated with a nonnegative *weight* $f(G)$ that, roughly speaking, measures how well G fits the given data. How the weight are obtained from the data and background knowledge varies depending on the adopted learning paradigm and performance measures [Koller and Friedman, 2009, Ch. 18]. Important for our purposes is that the commonly used weight functions are *modular*, i.e., they decompose into a product of vertex-wise weights:

$$f(G) = \prod_{v \in V} f_v(G_v).$$

We consider three different ways to formulate structure learning as an algorithmic problem. To this end, let $\mathcal{D}$ denote the set of DAGs on the vertex set V. We assume that the values $f_v(G_v)$, for all vertices v and their possible parent sets G_v, have been precomputed and given as input. We assume the *non-zero representation*, in which a weight for a parent set is (explicitly) given only when it is non-zero. Furthermore, we assume that the size of this representation is polynomial in the number of vertices.

First, consider maximizing the weight function:

BAYESIAN NETWORK STRUCTURE LEARNING (BNSL)
Objective: Compute $\max_{G \in \mathcal{D}} f(G)$.

While our formulation only asks for the maximum weight, the algorithms typically also yield a maximizing DAG. To conform to the common nomenclature used in literature, we retain the term "learning" in the problem name. Sometimes BNSL is stated as maximization of an additively decomposable scoring function; one obtains this equivalent form by considering a logarithm of the weight function.

Second, consider weighted counting of DAGs:

BAYESIAN NETWORK STRUCTURE COUNTING (BNSC)
Objective: Compute $\sum_{G \in \mathcal{D}} f(G)$.

In applications, $f(G)$ may equal an unnormalized posterior probability of G, in which case the sum equals the normalizing constant. More generally, $f(G)$ may equal a product of the posterior probability of G and some other function $h(G)$, e.g., an indicator function telling whether or not some edges of interest are present in G [Friedman and Koller, 2003, Tian and He, 2009]. Then the sum equals the posterior expectation of $h(G)$, or the posterior probability of the event of interest in the case of an indicator function.

Third, consider sampling DAGs with probabilities proportionally to the weights:

BAYESIAN NETWORK STRUCTURE SAMPLING (BNSS)
Objective: Sample $G \in \mathcal{D}$ with $\Pr(G) \propto f(G)$.

Here $f(G)$ typically is an unnormalized posterior probability of G, albeit one could also consider sampling from distributions that are biased, e.g., towards some graph features in interest; cf. indicators of edge sets discussed above.

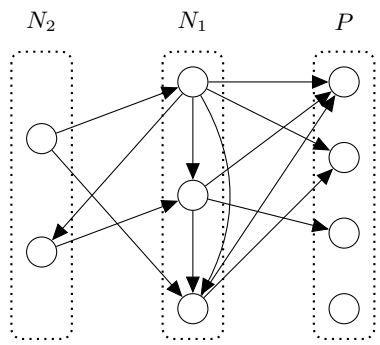

Figure 1: One network structure partitioned into a core and a periphery with a vertex cover N_1 of size $k = 3$.

2.1 STRUCTURAL CONSTRAINTS

In this paper, we focus on the constraint studied by Korhonen and Parviainen [2015], the *vertex cover number of the moralized graph*. Let G_{M} be the undirected graph obtained by *moralizing* $G = (V, E)$, meaning that we add an edge between u and v if there is a directed edge between them or they form a *v-structure*: both of them are parents of the same vertex without being connected by an edge.

With a slight abuse of terminology, G is said to have a *vertex cover* S of size k if all edges in G_{M} have at least one endpoint in S. The *vertex cover number* $\tau(G)$ of a graph is the size of the smallest vertex cover of G_{M}. We denote the set of DAGs G with $\tau(G) \leq k$ by $\mathcal{V}_k$ and the constrained variants of the problems restricted to this set by $\mathcal{V}_k$-BNSL, $\mathcal{V}_k$-BNSC, and $\mathcal{V}_k$-BNSS.

3 MAXIMIZATION

Observe that the vertices in the vertex cover can have at most one parent outside the vertex cover: Otherwise, it would no longer be the vertex cover of the moralized graph as the parents would be connected. Using this idea, Korhonen and Parviainen [2015] proposed the following algorithm for solving the problem: Brute force search over all possible vertex covers N_1 of the graph and the set of their parents N_2. These two sets are the *core* of the structure, and the remaining vertices P are the *periphery* (see Fig. 1). Importantly, the core and the periphery can be optimized independently.

For the core, they use the exponential-time algorithm to find its optimal structure in roughly $2^{|N_1|+|N_2|} \leq 2^{2k}$ operations (see, e.g., Silander and Myllymäki [2006]). The vertices in the periphery are then connected to their best parent sets in N_1, which is achieved efficiently with precomputation: For each node and a set S of possible parents of size at most k, find the optimal parent set for the node among the subsets of S. The total running time of their algorithm is $(2\mathrm{e}n/k)^{2k} n^{O(1)}$ where $n \coloneqq |V|$. The time complexity has an atypical form because the sets N_1 and N_2 are unordered, meaning that there are roughly $(n^k/(k!))^2$ possibilities for the sets. We improve this complexity to $3^k n^{k+O(1)}$, achieving nearly quadratic speedup in n:

Theorem 1. $\mathcal{V}_k$-BNSL *can be solved in time* $3^k n^{k+O(1)}$.

Proof. Unlike in the algorithm of Korhonen and Parviainen [2015], we only brute force over the set N_1 and its orders $(v_1, v_2, \ldots, v_k)$ in roughly n^k operations. Then, we complete the structure of the core using dynamic programming that decides how the vertices are distributed between N_2 and P. The dynamic programming computes the optimal structure for the ordered vertices N_1 and the first i of the (arbitrarily ordered) remaining vertices $w_1, w_2, \ldots, w_{n-k} \in V$ such that some subset of N_1 has parents.

More formally, let $g_v(S)$ be the highest weight for v from a parent set belonging to S and similarly $g_v(S, u)$ the highest weight such that the parent set is a subset of $S \cup \{u\}$ and contains u. Then, we initialize

$$d^{\mathrm{opt}}_{\emptyset,0} \coloneqq \prod_{j=1}^{k} g_{v_j}(v_{1:(j-1)}), \quad d^{\mathrm{opt}}_{S,0} \coloneqq 0 \text{ for all } S \subseteq v_{1:k}$$

with $v_{1:j}$ denoting the sequence $v_1, v_2, \ldots, v_j$. In other words, $d^{\mathrm{opt}}_{\emptyset,0}$ is the highest scoring DAG of $v_{1:k}$ whose one of the linear extensions is $v_1 < v_2 < \cdots < v_n$, that is, there is no path from v_{i+1} to v_i. As no other vertices have been added yet, the vertices in N_1 lack parents outside itself.

If a vertex w is assigned to the periphery, it always chooses the best parent set from N_1. Otherwise, it is the parent of some subset T of the vertices in N_1, in which case each such vertex v picks the best parent set from the union of its predecessors and w. To prevent cycles, the parent set choices must respect the assumption that $v_1 < v_2 < \cdots v_n$. This is achieved by forcing the parents of w to appear before the first vertex that w is a parent of in the order of the vertices of N_1. Therefore, we define $d^{\mathrm{opt}}_{S,i}$ as follows:

$$d^{\mathrm{opt}}_{S,i} \coloneqq \max_{T \subseteq S} d^{\mathrm{opt}}_{S \setminus T, i-1} \cdot r(w_i, T), \tag{1}$$

where $r(w_i, T)$ is the factor by which the optimal weight changes if w_i is a parent of $T \subseteq N_1$:

$$r(w_i, T) \coloneqq g_{w_i}\big(\{v_j \mid v_j < T\}\big) \cdot \prod_{v_j \in T} \frac{g_{v_j}\big(v_{1:(j-1)}, w_i\big)}{g_{v_j}\big(v_{1:(j-1)}\big)}$$

with $v_j < T$ if $j < \ell$ for all $v_\ell \in T$.

Now, the highest weight of a DAG that respects the order of the vertices of N_1 can be found as $\max_S d^{\mathrm{opt}}_{S,n-k}$. Iterating over all orders of N_1 then covers the set of all DAGs. We needed to consider 3^k sets S and T over the execution of the algorithm for each of the $n - k$ dynamic programming layers, and there are $O(n^k)$ options for the ordered elements of N_1, giving the total running time $3^k n^{k+O(1)}$. □

Consider instead the more common (equivalent) problem of optimizing the logarithm of the weight. Then, Equation 1 can be interpreted as subset convolution over the max–sum semiring, for which fast algorithms exist assuming that the values are integers with a bounded absolute value W [Björklund et al., 2007]. In such a case, the values $d^{\text{opt}}_{S,i}$ are computable in roughly $2^k W$ operations for each fixed i.

Corollary 2. $\mathcal{V}_k$-BNSL *can be solved in time* $2^k W n^{k+O(1)}$ *if the logarithms of the input weights, in some base, are integers with absolute value at most* W.

4 COUNTING

We start by showing that weighted counting of DAGs is hard. We also prove that the parameterized version is hard, but that there exists at least a simple albeit slow algorithm for the problem that outperforms the nonparameterized algorithm.

4.1 GENERAL CASE IS #P-HARD

We prove that the problem is #P-*hard*: at least as hard as counting the number of satisfying assignments for a SAT formula. To show this, we construct a reduction from a #P-*complete* problem MONOTONE 2-SAT [Valiant, 1979], where we are given a conjunction of m clauses of two positive literals, $(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_m \vee b_m)$ with $a_i, b_i \in \{x_1, x_2, \ldots, x_n\}$. The objective is to count the number of its satisfying assignments.

Theorem 3. *The* BNSC *problem is* #P-*hard.*

Proof. Consider the following reduction into counting the total weight of all DAGs visualized in Figure 2.

We have $n + 2m + 1$ vertices: n vertices $X_1, X_2, \ldots, X_n$ corresponding to the variables, m vertices $C_1, C_2, \ldots, C_m$ corresponding to the clauses, one additional node B_i for each clause C_i, and a single node D for choosing a subset of positive variables. If D is a parent of X_j, then we interpret that as if the variable x_j is set to be true, and false otherwise.

Let V_i be the set of two variables in the clause $a_i \vee b_i$, and define the following weights:

- $f_{C_i}(V_i) = 1$;
- $f_{B_i}(\{C_i\}) = M$ with $M \coloneqq 2^{n+1} - 1$;
- $f_{B_i}(\emptyset) = 1$;
- $f_D(\{B_1, B_2, \ldots, B_m\}) = 1$;
- $f_{X_j}(\{D\}) = f_{X_j}(\emptyset) = 1$.

First, consider a satisfying assignment of variables (x_j). The parent set of each B_i has to be empty since otherwise there would be a cycle in the graph: there is a directed path from B_i to C_i through some X_j. Thus, the weight is 1.

If the assignment (x_j) does not satisfy all clauses, each unsatisfied clause C_i contributes a factor $(M + 1)$ to the weight of the assignment: vertex B_i can either have C_i as a parent or have no parents, as there is no path from D to C_i.

Now, the total weight modulo $M + 1$ gives the number of satisfying assignments as any unsatisfying one is removed from the sum and there are exactly 2^n assignments. □

With some slight modifications, we get even stronger results:

Corollary 4. BNSC *is* #P-*hard even if each vertex can have at most* 2 *parents and the weights are either* 0 *or* 1.

Proof. In the original construction, D has exactly m parents. We fix this by instead constructing a binary tree of at most $2m$ vertices with the leaf layer connected to the clauses C_i.

For achieving binary weights, we replace the possible edge between each B_i and C_i by an *amplifier* that produces a big number if the edge could be there: For each node B_i, build a binary tree with $n+1$ leaf nodes with edges directed towards B_i. Then, let each leaf node have parent sets $\{C_i\}$ and $\emptyset$ with weight 1. Again, none of them can have parents for a satisfying assignment, resulting in a weight of 1. However, each of them has two choices for an unsatisfying assignment, producing a factor of 2^{n+1} to the sum of weights. Again, we take the result modulo $M + 1$. □

4.2 PARAMETERIZED CASE IS #W[1]-HARD

While constructing the same structure with different partitions into N_1, N_2, and P is not an issue when searching for the best structure, it certainly is in counting the sum of the weights of all valid DAGs. Consider, for example, a network of four nodes v_1, v_2, v_3, and v_4 such that v_1 and v_2 are the parents of v_3, which is the parent of v_4. What should be its canonical representation? If we fix $k = 2$, then v_3 has to be in N_1. However, now v_1 and v_2 cannot both be in N_2 nor in N_1 due to the v-structure and the parameter k. Further complicating the matter, we can put v_4 to P or to N_2.

We proceed to show that $\mathcal{V}_k$-BNSC is #W[1]-hard, meaning that no algorithm of time complexity $f(k)n^{O(1)}$ is likely to exist under typical complexity theoretical assumptions.

Theorem 5. $\mathcal{V}_k$-BNSC *is* #W[1]-*hard even if each vertex can have at most* 2 *parents.*

Proof. The result follows rather directly from the W[1]-hardness result of Korhonen and Parviainen [2015] for $\mathcal{V}_k$-BNSL. Counting the cliques of size k is a #W[1]-complete problem [Flum and Grohe, 2004], and we show how to adapt $\mathcal{V}_k$-BNSC into counting them.

For each edge $\{v, w\}$ of the input graph $G = (V, E)$ for counting cliques, add a new *edge vertex* u whose parent set

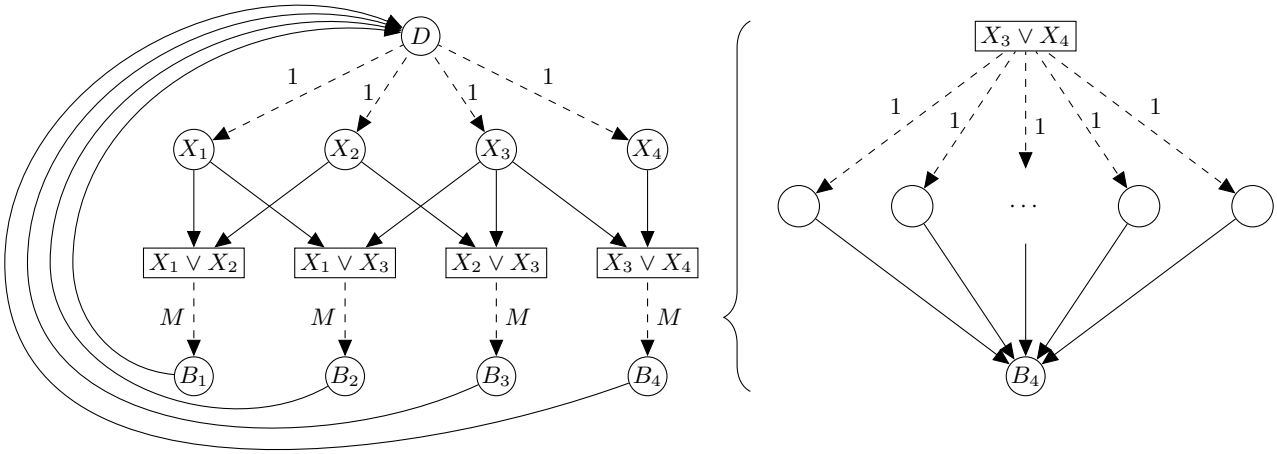

Figure 2: An example reduction from a MONOTONE 2-SAT instance to $\mathcal{V}_k$-BNSC. Solid edges are always on (weight 1, other choices have weight 0) while dashed edges are either on (weight written next to edge) or off (weight 1). Each M-weighted edge can be replaced with the binary-weighted component on the right.

$\{v, w\}$ has an indeterminate weight x, an empty parent set has weight 1, and the weights are 0 otherwise. For vertices in V, let only the empty parent set have weight 1.

Now, assume that the vertices of N_1 form a clique of size k in G. Then, there are exactly $k(k-1)/2$ edge vertices who can have a weight of x. Assuming all of their weights are x, then the remaining vertices need to be parentless for the graph to achieve a positive weight. Additionally, the number of ways the vertices can be distributed between P and N_2 is

$$\sum_{s=0}^{k} \binom{|V| + |E| - k}{s}.$$

On the other hand, a set N_1 that is not a k-clique (for $k > 3$) cannot achieve a weight with a factor $x^{k(k-1)/2}$: If N_1 contains a edge vertices, then there can be at most

$$\binom{|N_1| - a}{2} + a = \binom{|N_1|}{2} + \frac{a^2 - (2 \cdot |N_1| - 3)a}{2}$$

edge vertices with a parent set of value x. The rightmost summand is nonnegative only after $a \geq 2 \cdot |N_1| - 3$, but on the other hand $a \leq |N_1|$, meaning that the summand can be nonnegative only if $|N_1| \leq 3$. Thus, there are fewer than $\binom{k}{2}$ parent sets of value x when $k > 3$ and N_1 is not a k-clique of size k.

Computing the sum $\binom{k}{2} + 1$ times over different values of x enables us to discover the coefficient of the term $x^{k(k-1)/2}$ using polynomial interpolation. This is then divided by the constant number of ways of distributing the vertices between P and N_2 to obtain the number of k-cliques in G. □

Curiously, the maximization variant is known to be W[2]-hard by a reduction from SET COVER [Grüttemeier and Komusiewicz, 2022], but transforming that proof to show #W[2]-hardness of $\mathcal{V}_k$-BNSC appears challenging due to having to count each set cover only once.

4.3 EXACT COUNTING

To prevent duplicate counting, we need to determine a canonical decomposition into the sets N_1, N_2, and P for each valid structure. We achieve this by demanding that the core vertices must have children:

Definition 1. *A directed acyclic graph G with $\tau(G) \leq k$ and a partition into sets N_1, N_2, and P are called a* parent decomposition *if all vertices in N_1 and N_2 have a child. Furthermore, we require that either $|N_1| = k$ or $N_2 = \emptyset$.*

The latter constraint simplifies later proofs. Next, we show each valid DAG has such a decomposition and bound the number of ways of decomposing a DAG (used for sampling).

Lemma 6. *Each directed acyclic graph G with $\tau(G) \leq k$ has at least one and at most 2^k parent decompositions. The upper bound is tight up to a polynomial factor in k and is achieved by a core that is a chain of $3k/2$ vertices.*

Proof. Take any DAG G with $\tau(G) \leq k$ and its partition into the sets N_1, N_2, and P. Note that any childless vertex in N_2 can be moved to P. Similarly, any childless vertex in N_1 can be moved to P if its parents in N_2 are moved to N_1. As there can be at most one parent in N_2, the size of the set N_1 does not increase, keeping it as a valid vertex cover.

If $|N_1| \neq k$, we can always move vertices from N_2 to N_1 until it becomes full or N_2 is empty.

Each parent decomposition of a DAG has the same set of core vertices $N_1 \cup N_2$ but distinct sets N_2. Therefore, we prove the upper bound by bounding the number of possible sets N_2 for any set of core vertices. Furthermore, we show that a *chain* — a directed path — has the greatest number of such sets. First, note that N_2 has to be an independent set in the moralized graph and cannot contain core vertices with children in P. We proceed to restrict the graph structure and prove with injections that the maximum number of independent sets will not decrease.

Pick any independent set in the moralized core. That set remains independent even if we removed all but one outgoing edge from each vertex before moralization. Thus, we have restricted ourselves into a directed forest.

Now, consider any tree in the forest. If it is not a chain, there always exists a vertex v whose ancestors would form multiple chains if v were removed. We show that detaching all but one of these chains and joining them together into a longer chain has at least as many independent sets in the moralized graph using the crucial property that v and its parents form a clique after moralization. We define the injection as follows: If none of the parents of v is in the independent set, then the resulting independent set will remain the same in the longer chain. Otherwise, replace all descendants of the parent of v in the independent set by their children. This works because the parents of v separate the subchains from each other and only one of them can be in the independent set.

Finally, observe that each chain ends in a vertex with children in the periphery since we required the partition to be a parent decomposition. Such vertices cannot be in N_2, meaning that concatenating the chains cannot decrease the number of parent decompositions. Thus, a single chain core has the largest number of them. If the chain consists of less than k vertices, then N_2 is empty by the definition of a parent decomposition. Otherwise, the chain has c vertices and there are $\binom{k}{c-k}$ ways to choose an independent set of size $c-k$. This is maximized when $c = 3k/2$ and approaches asymptotically $2^k/\sqrt{k\pi/2}$ by Stirling's approximation. □

To summarize our approach, we iterate over all cores and its structures, connect N_1 to the periphery, and verify that

i it is a valid DAG,

ii each core vertex has a child, and

iii the choice of the set N_2 is lexicographically the smallest for that core (with a fixed vertex ordering).

Before going through with the detailed proof of the algorithm, we need a data structure for efficiently computing the weights from connecting N_1 to the periphery.

Lemma 7. *Given a partition into sets N_1, N_2, and P, a data structure can be built in time $4^k n^{O(1)}$ that allows querying the total weight of parent set choices for P with exactly the subset $S \subseteq N_1$ having children in P in time $n^{O(1)}$.*

Proof. Index vertices in $P = \{p_1, p_2, \ldots, p_{n-|N_1|-|N_2|}\}$ arbitrarily. We build a dynamic programming table $d^{\text{per}}_{S,i}$ describing the total weight of parent set choices of $p_{1:i}$ with exactly vertices in $S \subseteq N_1$ having children in P. The table is initialized with $d^{\text{per}}_{\emptyset,0} := 1$ and has the recurrence relation

$$d^{\text{per}}_{S,i} := \sum_{T \subseteq S} \sum_{\substack{U \subseteq S \\ U \supseteq S \setminus T}} d^{\text{per}}_{U,i-1} \cdot f_{p_i}(T).$$

Here, U is the set of vertices in N_1 that $p_{1:(i-1)}$ cover and T is the parent set of p_i.

To simplify notation, we let $d^{\text{per}}_S := d^{\text{per}}_{S,n-|N_1|-|N_2|}$ for all S. This is the answer to the query for a given S. □

We may now analyze the total complexity of our algorithm:

Theorem 8. $\mathcal{V}_k$-BNSC *can be solved in time* $2^{\binom{2k}{2}} 12^k n^{2k+O(1)}$.

Proof. Assume that we have already partitioned the vertices into N_1, N_2, and P. We start by iterating over all permutations of $N_1 \cup N_2$ and use that as a topological ordering for our core DAG. Then, we iterate over parent sets of core vertices respecting that topological ordering. Finally, we verify in polynomial time that the topological ordering used to generate the DAG is the lexicographically smallest one obtainable from that DAG. This generation of all core DAGs requires roughly $(2k)!2^{\binom{2k}{2}}$ work, matching the number of DAGs up to a single exponential factor [Stanley, 1973].

Next, we need to assign edges from the core to the periphery. To this end, iterate over subsets S of N_1 that should have children in P. We consider all such parent set assignments simultaneously by making use of the data structure of Lemma 7. Consider the graph resulting from any one of them. Any such graph clearly satisfies the property i. Property ii holds if vertices in N_2 and $N_1 \setminus S$ have a child in the core, which is checkable in polynomial time. Finally, we need to verify that N_2 is the lexicographically smallest possible set for the given structure. The set N_2 needs to be an independent set in the moralized graph without children in P. To check this, we moralize the core and iterate over independent subsets T of $N_1 \setminus S$. If there is a subset of N_2 of size $|N_2| - |T|$ that is independent of T, then T together with those vertices is also a valid option for the set N_2. Repeating this for all T allows us to find the lexicographically smallest set. If all properties hold, then we add the weight of the core multiplied by the weight of parent set choices for P such that exactly S has children in P to the total weight. Overall, there are 3^k choices for the sets S and T.

Finally, there are $\binom{n}{k}\binom{n-k}{k}$ unordered sets N_1 and N_2 over which we need to iterate. Thus, the total running time is $2^{\binom{2k}{2}}12^k n^{2k+O(1)}$. □

5 SAMPLING

The exact counting algorithm is unfortunately doubly exponential to avoid duplicate counting, but can we do better in estimating the sum of weights? It is a common phenomenon that approximating is computationally simpler than exact counting (consider, e.g., estimating the permanent of a matrix [Jerrum et al., 2004]). As counting and sampling objects with probabilities proportional to their weights are inherently connected, we approach the issue through sampling.

We develop a sampling method for network structures using *rejection sampling* in which we sample structures from a superset and then check whether they should be *accepted* or *rejected*. More precisely, we let valid structures appear multiple times in the superset and demand that exactly one of the duplicates should be accepted for each structure. This ensures the correct distribution of accepted samples and yields an acceptance probability that is the ratio of the total weight of the structures and the superset. Additionally, the sum of the weights is estimable by multiplying the empirical acceptance rate by the weight of the superset.

The key component of our algorithm is the fact of Lemma 6 that each valid DAG has at most 2^k parent decompositions. We use this property to first compute an approximation of the sum of the weights that is at most 2^k times greater than the exact sum. For this purpose, we iterate over all partitions of vertices into N_1, N_2, and P to compute the sum of weights of network structures that have a parent decomposition consistent with the partition. In addition, we further separate the structures by considering the unique sets $S \subseteq N_1$ that are the sinks of the subgraph induced by the core, because they need children in the periphery.

Before continuing, we need one more tool to aid dynamic programming: The *sink-layering* $L = (L_1, L_2, \ldots, L_\ell)$ of a DAG is an ordered partition of the vertices such that the last layer L_ℓ contains the sink vertices, $L_{\ell-1}$ the sinks after removing the vertices in L_ℓ, and so on. More formally, L_k comprises of the sinks of the subgraph induced by $V \setminus (L_{k+1} \cup L_{k+2} \cup \cdots \cup L_\ell)$. This structure lets us to go through the core layer by layer while only maintaining few subsets of vertices: the previous layer, used vertices, and the guess for the current layer. The strategy has been applied succesfully on sampling and counting DAGs by, for example, Kuipers and Moffa [2015] and Talvitie et al. [2019].

To efficiently utilize sink-layerings, we require a data structure that allows us to query the total weight of parent assignments for the vertices of the current layer $C = L_i$ such that the union of their parents covers the previous layer $F = L_{i-1}$ and is a subset of vertices $U = \bigcup_{j=1}^{i-1} L_j$ that have already been used. We denote this sum by $W(C, F, U)$.

Lemma 9. *The values of $W(C, F, U)$ can be precomputed in time $O\left(8^{2k}\right)$ for a set of core vertices $N_1 \cup N_2$.*

Proof. We iterate over all $O(4^{2k})$ choices for C, F, and U. First, initialize $W_0(C, F, U) = 0$ for all $F \subseteq U$ with the exception of $W_0(C, \emptyset, U) = 1$. For $i > 0$, we index the set $C = \{v_1, v_2, \ldots, v_{|C|}\}$ arbitrarily, and compute $W_i(C, F, U)$ as the sum

$$\sum_{T \subseteq F} W_{i-1}(C, T, U) \cdot \tilde{f}_{v_i}\big(F \setminus T \subseteq G_{v_i} \subseteq U\big),$$

with $\tilde{f}_v(\psi(G_v))$ being the sum of weights of parent sets G_v that satisfy the condition $\psi(G_v)$. In other words, for $v_1, v_2, \ldots, v_i$ to cover a set F of vertices, the parents of $v_1, v_2, \ldots, v_{i-1}$ cover a subset of F and v_i the rest (and possibly some vertices that are already covered or outside F). Finally, we let $W(C, F, U) \coloneqq W_{|C|}(C, F, U)$.

We can precompute $\tilde{f}_v$ in $O(4^{2k})$ time before starting to compute the values of W_i, and each $W_i(C, F, U)$ considers $2^{|F|} \leq 2^{2k}$ subsets T, proving the time complexity. □

Lemma 10. *Given a partition into sets N_1, N_2, and P in addition to a subset S of N_1 and precomputed values of W, we can compute the total weight of valid parent set choices for the core such that exactly the vertices in S are the sinks of the core in time $O(4^{2k})$.*

Proof. Inspired by Talvitie et al. [2019], we proceed layer by layer from L_1 towards L_ℓ. No layer is empty, so we need to consider at most $2k$ layers. Let $\overline{S}$ be the set $(N_1 \cup N_2) \setminus S$ and $d^{\text{core}}_{F,U}$ denote the total weight of DAGs with a vertex set U such that their sinks are F. We initialize

$$d^{\text{core}}_{U,U} \coloneqq \prod_{v \in U} f_v(\emptyset)$$

for all $U \subseteq \overline{S}$: the vertices in in the first layer lack parents.

For the following layers, we guess the previous layer, the current layer, and the set of used vertices. This results in the following recurrence for all $U, C \subseteq \overline{S}$ with $U \cap C = \emptyset$:

$$d^{\text{core}}_{U \cup C, C} \coloneqq \sum_{F \subseteq U} d^{\text{core}}_{U,F} \cdot W(C, F, U)$$

Finally, we need to include S on the last layer by defining

$$d^{\text{core}}_{(N_1 \cup N_2), S} \coloneqq \sum_{F \subseteq \overline{S}} d^{\text{core}}_{\overline{S},F} \cdot W(S, F, \overline{S}).$$

The variable $d^{\text{core}}_{(N_1 \cup N_2), S}$ now contains the desired value. There are roughly 4^{2k} choices for the sets C, F, and U, proving the complexity. □

Theorem 11. *A 2^k-approximation of the solution to $\mathcal{V}_k$-*BNSC *can be computed in time $(4\mathrm{e}n/k)^{2k}n^{O(1)}$.*

Proof. We calculate the 2^k-approximation by iterating over the parent decompositions and the sets S of core vertices without children in the core. This corresponds to the sum

$$\mathrm{UB} := \sum_{(N_1,N_2,P)} \sum_{S\subseteq N_1} d^{\mathrm{core}}_{(N_1\cup N_2),S} \sum_{T\supseteq S} d^{\mathrm{per}}_T .$$

By Lemma 6, each valid DAG has at least one and at most 2^k parent decompositions, proving the accuracy guarantee.

The rightmost sum can be precomputed beforehand for all S. Then, we compute the values of W over all sets of vertices of size at most $2k$. By Lemma 9, this takes $O\left(8^{2k}\right)$ time, and we consider roughly $\binom{n}{2k} \approx n^{2k}/(2k)!$ sets. By Stirling's approximation, this takes $(4\mathrm{e}n/k)^{2k}\, n^{O(1)}$ time in total.

Assuming the values of W are already precomputed, we can find the values of d^{core} in time 4^{2k} for each of the roughly $\binom{n}{k}\binom{n-k}{k} \approx n^{2k}/(k!)^2$ sets N_1 and N_2. Thus, the overall running time becomes $(4\mathrm{e}n/k)^{2k}n^{O(1)}$. □

We can now prove our main result on sampling. Hereinafter, write $\operatorname{par} G$ for the number of parent decompositions of G.

Theorem 12. $\mathcal{V}_k$-BNSS *can be solved with preprocessing time $(4\mathrm{e}n/k)^{2k}n^{O(1)}$ and expected sampling time $2^k n^{O(1)}$ using a randomized algorithm.*

Proof. Nearly all our data structures use only simple summation over sets of vertices with constraints. This allows the use of standard stochastic backtracking routines to sample from the structures: If the summands are saved in an array, we can run binary search (after trivial preprocessing) to move backwards in the dynamic programming tables to deduce the structure of the graph. As the sizes of the tables are exponential in k, the running time of the binary search is only linear in k and logarithmic in n.

First, we sample the partition (N_1, N_2, P) of the vertices of the DAG and the set S, and then proceed on figuring out the structure of the core and the periphery. At this point, the probability of drawing G is proportional to $f(G) \cdot \operatorname{par} G$.

Finally, we need to prevent sampling the same structure in multiple ways. This is solved with rejection sampling by accepting only one parent decomposition for each DAG. Similar to exact counting, we accept the parent decomposition if and only if N_2 is the lexicographically smallest set possible among all parent decompositions of G. We iterate over all subsets T of N_1 that are independent in the moralized graph, and find $|N_2| - |T|$ lexicographically smallest vertices in N_2 that are independent of T. If the current independent set N_2 is the lexicographically smallest obtainable set, accept the sample, and otherwise reject it. Thus, the acceptance probability of a DAG G is $1/\operatorname{par} G \geq 2^{-k}$. □

Overall, the probability of accepting a sample is then

$$\frac{\sum_{G\in\mathcal{V}_k} f(G)}{\sum_{G\in\mathcal{V}_k} f(G)\cdot \operatorname{par} G},$$

where the normalizing constant is already known, UB. Therefore, rejection sampling also enables us to estimate the sum of weights in arbitrary precision by approximating the acceptance ratio empirically and multiplying by UB.

We say that an algorithm computes an (ϵ,δ)-approximation of a quantity if its relative error is at most ϵ with probability at least $1-\delta$. By a result of Dagum et al. [2000], we have an (ϵ,δ)-approximation of the acceptance ratio immediately after getting $1+4(\mathrm{e}-2)(1+\epsilon)\epsilon^{-2}\ln(2/\delta)$ accepted draws. Thus, we have the following result:

Theorem 13. $\mathcal{V}_k$-BNSC *admits (ϵ,δ)-approximation in expected time $\left((4\mathrm{e}n/k)^{2k} + 2^k\epsilon^{-2}\ln\left(\delta^{-1}\right)\right)n^{O(1)}$.*

Proof. The preprocessing takes $(4\mathrm{e}n/k)^{2k}n^{O(1)}$ time, and sampling takes $2^k\epsilon^{-2}\ln\left(\delta^{-1}\right)n^{O(1)}$ time on average. □

6 CONCLUDING REMARKS

We advanced three problems on learning Bayesian network structures—maximization, weighted counting, and sampling—parameterized by the vertex cover number of the moralized DAG. First, we showed a nearly quadratic speedup on a previous maximization algorithm, thus significantly extending the scope of practical instances. Second, we presented the first parameterized algorithms for the sampling and counting variants, with polynomial running times for any constant value of the parameter. It remains an open question whether a quadratic speedup can be achieved also for these variants. We complemented our algorithmic results by complexity-theoretical hardness results on weighted counting both in the general and in the parameterized case.

The difficulties we encountered in avoiding duplicate counting raise an intriguing question: are there parameterized classes of DAGs where counting is significantly harder than maximization? One candidate parameter is the number of edges in the DAG, which renders the exponent of n independent of the parameter for maximization [Grüttemeier and Komusiewicz, 2022]; the problems resemble those of finding and counting paths of desired length, the latter variant known to be significantly harder [Chen and Flum, 2007]. Overall, counting and sampling have been studied little compared to the maximization variant, leaving numerous questions for future research.

References

Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset con-

volution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007.

Yijia Chen and Jörg Flum. On parameterized path and chordless path problems. In *Proceedings of the 22nd Annual IEEE Conference on Computational Complexity, CCC 2007*, pages 250–263. IEEE Computer Society, 2007.

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

Paul Dagum, Richard M. Karp, Michael Luby, and Sheldon M. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM J. Comput.*, 29(5):1484–1496, 2000.

Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Mach. Learn.*, 50(1-2): 95–125, 2003.

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

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

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

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

Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51 (4):671–697, 2004.

Mikko Koivisto and Antti Röyskö. Fast multi-subset transform and weighted sums over acyclic digraphs. In *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020*, volume 162 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *J. Mach. Learn. Res.*, 5: 549–573, 2004.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009. ISBN 978-0-262-01319-2.

Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 370–378. JMLR.org, 2013.

Janne H. Korhonen and Pekka Parviainen. Tractable Bayesian network structure learning with bounded vertex cover number. In *Advances in Neural Information Processing Systems 28, NIPS 2015*, pages 622–630, 2015.

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

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

David Madigan and Jeremy York. Bayesian graphical models for discrete data. *Int. Stat. Rev.*, 63:215–232, 1995.

Teppo Niinimaki, Pekka Parviainen, and Mikko Koivisto. Structure discovery in Bayesian networks by sampling partial orders. *J. Mach. Learn. Res.*, 17:57:1–57:47, 2016.

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

Tomi Silander and Petri Myllymäki. 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, 2006.

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

Richard P Stanley. Acyclic orientations of graphs. *Discrete Math.*, 5(2):171–178, 1973.

Topi Talvitie, Aleksis Vuoksenmaa, and Mikko Koivisto. 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, 2019.

Jin Tian and Ru He. Computing posterior probabilities of structural features in Bayesian networks. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009*, pages 538–547. AUAI Press, 2009.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.