Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs*

Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz
Institute of Theoretical Computer Science, University of Lübeck, Germany
{wienoebst,bannach,liskiewi}@tcs.uni-luebeck.de

Abstract

Counting and sampling directed acyclic graphs from a Markov equivalence class are fundamental tasks in graphical causal analysis. In this paper, we show that these tasks can be performed in polynomial time, solving a long-standing open problem in this area. Our algorithms are effective and easily implementable. Experimental results show that the algorithms significantly outperform state-of-the-art methods.

1 Introduction

Graphical modeling plays a key role in causal theory, allowing to express complex causal phenomena in an elegant, mathematically sound way. One of the most popular graphical models are directed acyclic graphs (DAGs), which represent direct causal influences between random variables by directed edges (Spirtes, Glymour, and Scheines 2000; Pearl 2009; Koller and Friedman 2009). They are commonly used in empirical sciences to discover and understand causal effects. However, in practice, the underlying DAG is usually unknown, since, typically, no single DAG explains the observational data. Instead, the statistical properties of the data are maintained by a number of different DAGs, which constitute a Markov equivalence class (MEC, for short). Therefore, these DAGs are indistinguishable on the basis of observations alone (Verma and Pearl 1990, 1992; Heckerman, Geiger, and Chickering 1995).

It is of great importance to investigate model learning and to analyze causal phenomena using MECs directly rather than the DAGs themselves. Consequently, this has led to intensive studies on Markov equivalence classes of DAGs and resulted in a long and successful track record. Our work contributes to this line of research by providing the first polynomial-time algorithms for *counting* and for *uniform sampling* Markov equivalent DAGs – two important primitives in both theoretical and experimental studies.

Finding the graphical criterion for two DAGs to be Markov equivalent (Verma and Pearl 1990) and providing the graph-theoretic characterization of MECs as so-called CPDAGs (Andersson, Madigan, and Perlman 1997) mark key turning points in this research direction. In particular, they have contributed to the progress of computational methods in this area. Important advantages of the modeling with CPDAGs are demonstrated by algorithms that learn causal structures from observational data (Verma and Pearl 1992; Meek 1995, 1997; Spirtes, Glymour, and Scheines 2000; Chickering 2002a,b); and that analyze causality based on a given MEC, rather than a single DAG (Maathuis, Kalisch, and Bühlmann 2009; van der Zander and Liśkiewicz 2016; Perkovic et al. 2017). Algorithms that ignore Markov equivalence may lead to incorrect solutions.

A key characteristic of an MEC is its size, i. e., the number of DAGs in the class. It indicates uncertainty of the causal model inferred from observational data and it serves as an indicator for the performance of recovering true causal effects. Moreover, the feasibility of causal inference methods is often highly dependent on the size of the MEC; e. g., to estimate the average causal effects from observational data for a given CPDAG, as proposed by Maathuis, Kalisch, and Bühlmann (2009), one has to consider all DAGs in the class. Furthermore, computing the size of a Markov equivalence class is commonly used as a subroutine in practical algorithms. For example, when actively designing interventions, in order to identify the underlying true DAG in a given MEC, the size of the Markov equivalence subclass is an important metric to select the best intervention target (He and Geng 2008; Hauser and Bühlmann 2012; Shanmugam et al. 2015; Ghassami et al. 2018, 2019).

The first algorithmic approaches for counting the number of Markov equivalent DAGs relied on exhaustive search (Meek 1995; Madigan et al. 1996) based on the graphical characterization of Verma and Pearl (1990). The methods are computationally expensive as the size of an MEC represented by a CPDAG may be superexponential in the number of vertices of the graph. More recently, He, Jia, and Yu (2015) proposed to partition the MEC by fixing root variables in any undirected component of the CPDAG. This yields a recursive strategy for counting Markov equivalent DAGs, which forms the basis of several "root-picking" algorithms (He, Jia, and Yu 2015; Ghassami et al. 2019; Ganian, Hamm, and Talvitie 2020). As an alternative approach, recent methods utilize dynamic programming on the clique tree representation of chordal graphs and techniques from intervention design (Talvitie and Koivisto 2019; AhmadiTeshnizi, Salehkaleybar, and Kiyavash 2020).

*Extended version of paper accepted to the Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-2021).
Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The main drawback of the existing counting algorithms is that they have exponential worst-case run time. Moreover, as our experiments show, the state-of-the-art algorithms (Talvitie and Koivisto 2019; Ganian, Hamm, and Talvitie 2020; AhmadiTeshnizi, Salehkaleybar, and Kiyavash 2020) perform inadequately in practice on a wide range of instances.

The main achievement of our paper is the first polynomial-time algorithm for counting and for sampling Markov equivalent DAGs. The counting algorithm, called *Clique-Picking*, explores the clique tree representation of a chordal graph, but it avoids the use of computationally intractable dynamic programming on the clique tree. The Clique-Picking algorithm is effective, easy to implement, and our experimental results show that it significantly outperforms the state-of-the-art methods. Moreover, we show that, using the algorithm in a preprocessing phase, uniform sampling of Markov equivalent DAGs can be performed in linear time.

We prove that our results are tight in the sense that counting Markov equivalent DAGs that encode additional background knowledge is intractable under standard complexity-theoretic assumptions. This justifies the exponential time approaches by Meek (1995) and Ghassami et al. (2019).

The next section contains preliminaries on graphs and MECs. In Sec. 3, we present the ideas of our novel approach, and Sec. 4 explains how to avoid overcounting using minimal separators. Section 5 contains our algorithm and in Sec. 6 we analyze its time complexity and formally present the main results of the paper. Finally, Sec. 7 shows our experimental results. Due to space constraints, proofs are relocated to the appendix. We provide short proof sketches for the most important results in the main text.

2 Preliminaries

A graph $G = (V_G, E_G)$ consists of a set of vertices V_G and a set of edges $E_G \subseteq V_G \times V_G$. Throughout the paper, whenever the graph G is clear from the context, we will drop the subscript in this and analogous notations. An edge $u - v$ is undirected if $(u, v), (v, u) \in E_G$ and directed $u \to v$ if $(u, v) \in E_G$ and $(v, u) \notin E_G$. Graphs which contain undirected and directed edges are called partially directed. Directed acyclic graphs (DAGs) contain only directed edges and no directed cycle. We refer to the neighbors of a vertex u in G as $N_G(u)$ and denote the induced subgraph of G on a set $C \subseteq V$ by $G[C]$. The graph union $G \cup H$ includes edges present in G or in H[1].

The *skeleton* of a partially directed graph G is the undirected graph that results from ignoring edge directions. A *v-structure* in a partially directed graph G is an ordered triple of vertices (a, b, c) which induce the subgraph $a \to b \leftarrow c$.

A *clique* is a set K of pairwise adjacent vertices. We denote the set of all maximal cliques of G by $\Pi(G)$. In a connected graph, we call a set $S \subseteq V$ an *a-b-separator* for two nonadjacent vertices $a, b \in V$ if a and b are in different connected components in $G[V \setminus S]$. If no proper subset of S separates a and b we call S a *minimal a-b-separator*. We say a set S is a *minimal separator* if it is a minimal a-b-separator for any two vertices[2]. We denote the set of all minimal separators of a graph G by $\Delta(G)$. An undirected graph is called *chordal* if no subset of four or more vertices induces an undirected cycle. For every chordal graph on n vertices we have $|\Pi(G)| \leq n$ (Dirac 1961). Furthermore, it is well-known that a graph G is chordal if, and only if, all its minimal separators are cliques.

A Markov equivalence class (MEC) consists of DAGs encoding the same set of conditional independence relations among the variables. Due to Verma and Pearl (1990), we know that two DAGs are Markov equivalent if, and only if, they have the same skeleton and the same v-structures. An MEC can be represented by a CPDAG (*completed partially directed acyclic graph*), which is the union graph of the DAGs in the equivalence class it represents. The undirected components of a CPDAG are *undirected and connected chordal graphs* (UCCGs) (Andersson, Madigan, and Perlman 1997).

An orientation of a partially directed graph G is obtained by replacing each undirected edge with a directed one. Such an orientation is called *acyclic* if it does not contain a directed cycle and *moral* if it does not create a new v-structure (sometimes called immorality). In the following, we will only consider acyclic moral orientations (AMOs). For a partially ordered graph G, we denote by $\mathrm{AMO}(G)$ the set of all AMOs and by $\#\mathrm{AMO}(G)$ the number of AMOs of G. In particular, if G is a CPDAG representing an MEC then $\#\mathrm{AMO}(G)$ is the size of the class. In this paper, we also refer to the *computational problem* of counting the number of AMOs for a given CPDAG as #AMO.

In case we have an induced subgraph $a \to b - c$ in a partially directed graph, the edge between b and c is oriented $b \to c$ in all AMOs. This is known as the first Meek rule (Meek 1995).

For a CPDAG G, the AMOs of each UCCG of G can be chosen independently of the other UCCGs and the directed part of G (Andersson, Madigan, and Perlman 1997). Thus,

$$\#\mathrm{AMO}(G) = \prod_{H \text{ is UCCG in } G} \#\mathrm{AMO}(H).$$

Hence, the problem #AMO of counting the number of DAGs in an MEC reduces to counting the number of AMOs in a UCCG (Gillispie and Perlman 2002; He and Geng 2008).

An AMO α of a graph G can be represented by a (not necessarily unique) linear ordering of the vertices. Such a *topological* ordering τ represents α if for each edge $u \to v$ in α, u precedes v in τ. We denote all topological orderings representing an AMO α of a graph G by $\mathrm{top}_G(\alpha) = \{\tau_1, \ldots, \tau_\ell\}$. Note that every AMO of a UCCG contains exactly one source vertex, i. e., a vertex with no incoming edges.

The s-orientation G^s of a UCCG G is the union of all AMOs of G with unique source vertex s. We view s-orientations from the equivalent perspective of being the union of all AMOs that can be represented by a topological ordering starting with s. The undirected components of G^s are UCCGs and can be oriented independently (He, Jia, and

[1] For example, the union of $a \to b \to c$ and $a \leftarrow b \to c$ is the graph $a - b \to c$.

[2] Observe that a minimal separator can be a proper subset of another minimal separator (for different vertex pairs a-b).

Yu 2015). This observation enables recursive strategies for counting AMOs: the "root-picking" approaches (He, Jia, and Yu 2015; Ghassami et al. 2019; Talvitie and Koivisto 2019; Ganian, Hamm, and Talvitie 2020) that pick each vertex s as source and recurse on the UCCGs of the s-orientation. Because these UCCGs can be oriented independently, the number of AMOs is obtained by alternately summing over the number of AMOs for each source vertex s and multiplying the number of AMOs for each independent UCCG.

3 Lexicographic BFS and AMOs

We introduce the core ideas of our algorithm for #AMO and a linear-time algorithm for finding the UCCGs of the s-orientations and their generalization, the $\pi(K)$-*orientations*. We do this by connecting AMOs with so-called *perfect elimination orderings* (PEOs). A linear ordering of the vertices is a PEO if for each vertex v, the neighbors of v that occur after v form a clique. A graph is chordal if, and only if, it has a PEO (Fulkerson and Gross 1965).

Lemma 1. *A topological ordering τ of the vertices of a UCCG G represents an AMO if, and only if, it is the reverse of a perfect elimination ordering.*

Perfect elimination orderings can be computed in linear time with the Lexicographic BFS algorithm (Rose, Tarjan, and Lueker 1976) to which we will refer as *LBFS*. A modified version of this algorithm is presented as Algorithm 1. When called with $K = \emptyset$ (and ignoring the lines 7-10), it coincides with a *normal* LBFS. The modifications and the meaning of K will become clear later on.

input : A UCCG $G = (V, E)$ and a clique $K \subseteq V$.
output: $\mathcal{C}_G(K)$.
1 $\mathcal{S} \leftarrow$ sequence of sets initialized with $(K, V \setminus K)$
2 $\tau \leftarrow$ empty list, $L \leftarrow \emptyset$
3 **while** $\mathcal{S}$ *is non-empty* **do**
4 $\quad X \leftarrow$ first non-empty set of $\mathcal{S}$
5 $\quad v \leftarrow$ arbitrary vertex from X
6 $\quad$ Add vertex v to the end of τ.
7 $\quad$ **if** v *is neither in a set in L nor in K* **then**
8 $\quad\quad L \leftarrow L \cup \{X\}$
9 $\quad\quad$ Output the undirected components of $G[X]$.
10 $\quad$ **end**
11 $\quad X \leftarrow X \setminus \{v\}$
12 $\quad$ Denote the current $\mathcal{S}$ by $(S_1, \ldots, S_k)$.
13 $\quad$ Replace each S_i by $S_i \cap N(v), S_i \setminus N(v)$.
14 $\quad$ Remove all empty sets from $\mathcal{S}$.
15 **end**

Algorithm 1: A modified version of the Lexicographic BFS (Rose, Tarjan, and Lueker 1976) for computing the set $\mathcal{C}_G(K)$. If the algorithm is executed with $K = \emptyset$, the algorithm performs a normal LBFS with corresponding traversal ordering τ, which is the reverse of a PEO.

LBFS runs in linear time (i. e., $\mathcal{O}(|V| + |E|)$) when $\mathcal{S}$ is implemented as a doubly-linked list with a pointer to each vertex and the beginning of each set. The algorithm can be viewed as a fine-grained graph traversal compared to classical breadth-first search (BFS), where the vertices are visited only by increasing distance to the start vertex. LBFS keeps this property, but introduces additional constraints on the ordering τ, in which the vertices are visited (τ is called an *LBFS ordering*). These constraints guarantee that τ is the reverse of a PEO. Hence, by Lemma 1, τ represents an AMO.

Corollary 1. *Every LBFS ordering τ of the vertices of a UCCG G represents an AMO.*

It holds even further that each AMO can be represented by at least one LBFS ordering.

Lemma 2. *Every AMO of a UCCG G can be represented by an LBFS ordering.*

Each LBFS ordering starts with a maximal clique, because as long as there is a vertex which can enlargen the current clique, the first set of $\mathcal{S}$ is made up solely of such vertices.

Lemma 3. *Every LBFS ordering starts with a maximal clique.*

These observations lead to the first idea in our algorithm for #AMO. We have seen that every AMO can be represented by an LBFS ordering (Lemma 2) and every LBFS ordering starts with a maximal clique (Lemma 3). It follows:

Corollary 2. *Every AMO can be represented by a topological ordering which starts with a maximal clique.*

This means that for us, it is *sufficient to consider topological orderings that start with a maximal clique*. Therefore, we generalize the definition of s-orientations: We consider permutations π of a clique K, as each $\pi(K)$ represents a distinct AMO of the subgraph induced by K.

Definition 1. *Let $G = (V, E)$ be a UCCG, K be a clique in G, and let $\pi(K)$ be a permutation of K.*

1. *The $\pi(K)$-orientation of G, also denoted $G^{\pi(K)}$, is the union of all AMOs of G that can be represented by a topological ordering beginning with $\pi(K)$.*
2. *Let G^K denote the union of $\pi(K)$-orientations of G over all π, i. e., let $G^K = \bigcup_\pi G^{\pi(K)}$.*
3. *Denote by $\mathcal{C}_G(\pi(K))$ the undirected connected components of $G^{\pi(K)}[V \setminus K]$ and let $\mathcal{C}_G(K)$ denote the undirected connected components of $G^K[V \setminus K]$.*

Figure 1 shows an example $\pi(K)$-orientation of G: For a graph G in **(a)**, a clique $K = \{1, 2, 3, 4\}$, and a permutation $(4, 3, 2, 1)$, graph $G^{(4,3,2,1)}$ is presented in **(c)**. It is the union of two DAGs which are AMOs of G, whose topological orderings begin with $4, 3, 2, 1$. The first DAG can be represented by topological ordering $4, 3, 2, 1, 5, 6, 7$ and the second one by $4, 3, 2, 1, 6, 5, 7$. In Fig. 1, we also compare the $(4, 3, 2, 1)$-orientation with an s-orientation, for $s = 4$, shown in **(b)**. The undirected components of the orientations are indicated by the colored regions. By orienting whole cliques at once, we get significantly smaller undirected components in the resulting $\pi(K)$-orientation than in the s-orientation (e. g., $\{5, 6\}$ compared to $\{1, 2, 3, 5, 6\}$). Finally, **(d)** illustrates graph $G^{\{1,2,3,4\}}$.

The undirected components of the $\pi(K)$-orientation are chordal graphs, which can be oriented independently, yielding the following recursive formula:

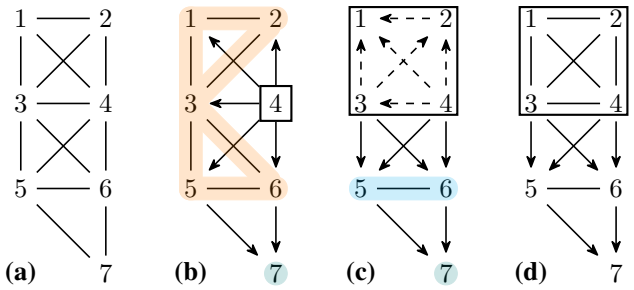


Figure 1: For a UCCG G in **(a)**, the figure shows $G^{(4)}$ in **(b)**, $G^{(4,3,2,1)}$ in **(c)**, and $G^{\{1,2,3,4\}}$ in **(d)**. The undirected components in $G^{(4)}$ and $G^{(4,3,2,1)}$ are indicated by the colored regions and the vertices put at the beginning of the topological ordering by a rectangle (all edges from the rectangle point outwards). Edges inside the rectangle in **(c)** are dashed, as they have no influence on the further edge directions outside the rectangle.

Lemma 4. *The undirected connected components in $\mathcal{C}_G(\pi(K))$ are chordal and it holds that:*

$$\#\text{AMO}(G^{\pi(K)}) = \prod_{H \in \mathcal{C}_G(\pi(K))} \#\text{AMO}(H).$$

The crucial observation is that the undirected components $\mathcal{C}_G(\pi(K))$ are independent of the permutation π. This means no matter how the vertices $\{1, 2, 3, 4\}$ are permuted, if the whole clique is put at the beginning of the topological ordering, no further edge orientations will be influenced. Informally, this is because all edges from the clique K to other vertices are directed outwards no matter the permutation π. We formalize this observation in the following:

Proposition 1. *Let G be a UCCG and K be a clique of G. For each permutation $\pi(K)$ it is true that all edges of $G^{\pi(K)}$ coincide with the edges of G^K, excluding the edges connecting the vertices in K. Hence, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(K)$ and it holds that:*

$$\sum_{\pi \text{ over } K} \#\text{AMO}(G^{\pi(K)}) = |K|! \times \prod_{H \in \mathcal{C}_G(K)} \#\text{AMO}(H).$$

This is the key property that allows us to efficiently deal with whole cliques at once, instead of considering single vertices one-by-one. As each permutation $\pi(K)$ represents a distinct AMO of $G[K]$, this formula indeed computes the number of AMOs, which can be represented by a topological ordering with clique K at the beginning. However, we are not done yet, as there are some further obstacles we need to overcome in order to obtain a polynomial-time algorithm for #AMO, which are dealt with in the following sections.

Before that, we leverage the connection between AMOs and LBFS orderings one more time, to propose a linear-time algorithm for computing $\mathcal{C}_G(K)$ – the full Algorithm 1. This algorithm performs an LBFS and, whenever a vertex could be picked for the first time, the corresponding first set in $\mathcal{S}$ is appended to L and the undirected components of the set are output (lines 7-10). For instance, in the example above, after the vertices in K are visited, we have $\mathcal{S} = (\{5, 6\}, \{7\})$. As $\{5, 6\}$ is currently the first set of $\mathcal{S}$ and vertices 5 and 6 are not in any set in L yet (L is still empty), the set $\{5, 6\}$ is appended to L and $5 - 6$ is output as an element of $\mathcal{C}_G(K)$.

Theorem 1. *For a chordal graph G and a clique K, Algorithm 1 computes $\mathcal{C}_G(K)$ in time $\mathcal{O}(|V| + |E|)$.*

Sketch of Proof. When a set is appended to L, each vertex of this set could have been the next chosen vertex in line 6. Thus, all edges between the vertices in a set in L may occur as either $u \to v$ (if u is chosen next) or as $u \leftarrow v$ (if v is chosen next) in an AMO having K at the beginning of the LBFS ordering. As a $\pi(K)$-orientation of G is the union of all corresponding AMOs, we have $u - v$.

If, on the other hand, u and v are neighbors but not in the same set in L, the edge between them is oriented $u \to v$ in G^K, assuming u is visited before v. This is due to an inductive argument by which $u \to v$ follows from iterative application of the first Meek rule. □

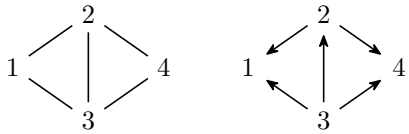
Both cases dealt with in the proof sketch can be seen in Fig. 1. The edge $5 - 6$ remains undirected as either vertex could be chosen first in Algorithm 1, while we have $5 \to 7$, because the first Meek rule applies to $\to 5 - 7$.

We note that this algorithm could also be used for finding the UCCGs of the s-orientations of a chordal graph in linear time, which improves upon prior work (He, Jia, and Yu 2015; Ghassami et al. 2019; Talvitie and Koivisto 2019).

4 Counting AM-Orientations with Minimal Separators and Maximal Cliques

Using the insights from the previous section, we would like to count the AMOs of a chordal graph G with the following recursive procedure based on Proposition 1: Pick a maximal clique K, consider all its permutations at once, and take the product of the recursively computed number of AMOs of the UCCGs of $\mathcal{C}_G(K)$. By Corollary 2, we will count every AMO in this way, if we compute the sum over all maximal cliques. Unfortunately, we will count some orientations multiple times, as a single AMO can be represented by multiple topological orderings. For instance, assume we have two maximal cliques K_1 and K_2 with $K_1 \cap K_2 = S$ such that $K_1 \setminus S$ is separated from $K_2 \setminus S$ in $G[V \setminus S]$. A topological ordering that starts with S can proceed with either $K_1 \setminus S$ or $K_2 \setminus S$ and result in the same AMO.

Example 1. Consider the following chordal graph (left) with maximal cliques $K_1 = \{1, 2, 3\}$ and $K_2 = \{2, 3, 4\}$. A possible AMO of the graph is shown on the right.



The AMO has two topological orderings: $\tau_1 = (3, 2, 1, 4)$ and $\tau_2 = (3, 2, 4, 1)$ starting with K_1 and K_2, respectively. Hence, if we count all topological orderings starting with K_1 and all topological orderings starting with K_2, we will count

the AMO twice. However, τ_1 and τ_2 have $3, 2$ as common prefix and $K_1 \cap K_2 = \{2, 3\}$ is a minimal separator of the graph – a fact that we will use in the following. ◇

Lemma 5. *Let α be an AMO of a chordal graph G and τ_1, τ_2 be two topological orderings that represent α. Then τ_1 and τ_2 have a common prefix $S \in \Delta(G) \cup \Pi(G)$.*

Note that this lemma implies that *all* topological orderings that correspond to an AMO have a common prefix, which is a minimal separator or maximal clique.

The combinatorial function ϕ, as defined below, plays an important role to avoid overcounting.

Definition 2. *For a set S and a collection R of subsets of S, we define $\phi(S, R)$ as the number of all permutations of S that do not have a set $S' \in R$ as prefix.*

Example 2. Consider the set $S = \{2, 3, 4, 5\}$ and the collection $R = \big\{\{2, 3\}, \{2, 3, 5\}\big\}$. Then $\phi(S, R) = 16$ since there are 16 permutations of $\{2, 3, 4, 5\}$ that neither start with $\{2, 3\}$ nor $\{2, 3, 5\}$ – e. g., $(3, 2, 4, 5)$ and $(2, 5, 3, 4)$ are forbidden as they start with $\{2, 3\}$ and $\{2, 3, 5\}$, respectively; but $(3, 5, 4, 2)$ is allowed. ◇

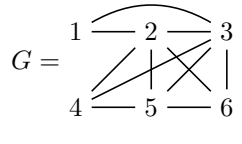
In this paper, we always consider sets $S \in \Delta(G) \cup \Pi(G)$ and collections $R \subseteq \Delta(G)$. Therefore, we can use the abbreviation $\phi(S) = \phi\big(S, \{\, S' \mid S' \in \Delta(G) \wedge S' \subsetneq S \,\}\big)$.

Proposition 2. *Let G be a UCCG. Then:*

$$\#\mathrm{AMO}(G) = \sum_{S \in \Delta(G) \cup \Pi(G)} \phi(S) \times \prod_{H \,\in\, \mathcal{C}_G(S)} \#\mathrm{AMO}(H).$$

Sketch of Proof. By Lemma 5, every AMO can be represented by a topological ordering that starts with a vertex set $S \in \Delta(G) \cup \Pi(G)$. The definition of $\phi(S)$ and an induction over the product yields the claim. □

Example 3. We consider the following chordal graph with two minimal separators and three maximal cliques:

$G =$ 

$$\Delta(G) = \big\{\{2, 3\}, \{2, 3, 5\}\big\}$$

$$\Pi(G) = \big\{\{1, 2, 3\}, \{2, 3, 4, 5\}, \{2, 3, 5, 6\}\big\}$$

To compute $\#\mathrm{AMO}(G)$ using Proposition 2, we need the following values. Note that the resulting subgraphs H are trivial, except for the case $S = \{2, 3\}$ and $S = \{1, 2, 3\}$. In these cases, we obtain the induced path on $\{4, 5, 6\}$, which has three possible AMOs.

$S \in \Delta(G) \cup \Pi(G)$	$\phi(S)$	$\prod_{H \in \mathcal{C}_G(S)} \#\mathrm{AMO}(H)$
$\{2, 3\}$	2	3
$\{2, 3, 5\}$	4	1
$\{1, 2, 3\}$	4	3
$\{2, 3, 4, 5\}$	16	1
$\{2, 3, 5, 6\}$	16	1

Using Proposition 2 we can compute $\#\mathrm{AMO}(G)$ as follows:

$$\#\mathrm{AMO}(G) = 2 \cdot 3 + 4 \cdot 1 + 4 \cdot 3 + 16 \cdot 1 + 16 \cdot 1 = 54.$$

We remark that we do *not* have discussed how to compute $\phi(S)$ yet – for this example, this can be done by naive enumeration. In general, however, this is a non-trivial task. We tackle this issue in the next section. ◇

5 The Clique-Picking Algorithm

In the previous section, we showed how to count AMOs by using minimal separators in order to avoid overcounting. It is rather easy to check that we can compute $\phi(S, R)$ in time *exponential* in $|R|$ using the inclusion-exclusion principle. However, our goal is *polynomial time* and, thus, we have to restrict the collection R.

Lemma 6. *Let S be a set and $R = \{X_1, \ldots, X_\ell\}$ be a collection of subsets of S with $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$. Then:*

$$\phi(S, R) = |S|! - \sum_{i=1}^{\ell} |S \setminus X_i|! \cdot \phi(X_i, \{X_1, \ldots, X_{i-1}\}).$$

Observe that this formula can be evaluated in polynomial time with respect to $|S|$ and ℓ, as all occurring subproblems have the form $\phi(X_i, \{X_1, \ldots, X_{i-1}\})$ and, thus, there are at most ℓ of them. The goal of this section is to develop a version of Proposition 2 based on this lemma.

To achieve this goal, we rely on the strong structural properties that chordal graphs entail: A *rooted clique tree* of a UCCG G is a triple (T, r, ι) such that (T, r) is a rooted tree and $\iota \colon V_T \to \Pi(G)$ a bijection between the nodes of T and the maximal cliques of G such that $\{\, x \mid v \in \iota(x) \,\}$ is connected in T for all $v \in V_G$. In slight abuse of notation, we denote, for a set $C \subseteq V_G$, by $\iota^{-1}(C)$ the subtree $\{\, x \mid C \subseteq \iota(x) \,\}$. We denote the children of a node v in a tree T by $\mathrm{children}_T(v)$. It is well-known that (i) every chordal graph has a rooted clique tree (T, r, ι) that can be computed in linear time, and (ii) a set $S \in V_G$ is a minimal separator if, and only if, there are two adjacent nodes $x, y \in V_T$ with $\iota(x) \cap \iota(y) = S$ (Blair and Peyton 1993).

We wish to interleave the structure provided by the clique tree with a formula for computing #AMO. For this sake, let us define the *forbidden prefixes* for a node v in a clique tree.

Definition 3. *Let G be a UCCG, $\mathcal{T} = (T, r, \iota)$ a rooted clique tree of G, v a node in T and $r = x_1, x_2, \ldots, x_p = v$ the unique r-v-path. We define the set $\mathrm{FP}(v, \mathcal{T})$ to contain all sets $\iota(x_i) \cap \iota(x_{i+1}) \subseteq \iota(v)$ for $1 \leq i < p$.*

Lemma 7. *We can order the elements of the set $\mathrm{FP}(v, \mathcal{T})$ as $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$.*

By combining the lemma with Lemma 6, we deduce that $\phi(\iota(v), \mathrm{FP}(v, \mathcal{T}))$ can be evaluated in polynomial time for nodes v of the clique tree. We are left with the task of developing a formula for #AMO in which all occurrences of ϕ are of this form. It is quite easy to come up with such formulas that count every AMO at least once – but, of course, we have to ensure that we count every AMO *exactly* once.

To ensure this property, we introduce for every AMO α a partial order $\prec_\alpha$ on the maximal cliques. Then we prove that there is a unique minimal element with respect to this order, and deduce a formula for #AMO that counts α only "at this minimal element". To get started, we need a technical definition and some auxiliary lemmas that give us more control over the rooted clique tree.

Definition 4. *An* S-flower *for a minimal separator S is a maximal set $F \subseteq \{\, K \mid K \in \Pi(G) \wedge S \subseteq K \,\}$ such that $\bigcup_{K \in F} K$ is connected in $G[V \setminus S]$. The* bouquet $\mathcal{B}(S)$ *of a minimal separator S is the set of all S-flowers.*

Example 4. The $\{2,3\}$-flowers of the graph from Example 3 are $\{\{1,2,3\}\}$ and $\{\{2,3,4,5\},\{2,3,5,6\}\}$. $\diamond$

Lemma 8. *An S-flower F is a connected subtree in a rooted clique tree (T,r,ι).*

Lemma 9. *For any minimal separator S, the bouquet $\mathcal{B}(S)$ is a partition of $\iota^{-1}(S)$.*

Since for a $S \in \Delta(G)$ the subtree $\iota^{-1}(S)$ of (T,r,ι) is connected, Lemma 8 and Lemma 9 give rise to the following order on S-flowers $F_1, F_2 \in \mathcal{B}(S)$: $F_1 \prec_T F_2$ if F_1 contains a node on the unique path from F_2 to the root of T.

Lemma 10. *There is a unique least S-flower in $\mathcal{B}(S)$ with respect to $\prec_T$.*

The lemma states that for every AMO α there is a flower F at which we want to count α. We have to be sure that this is possible, i. e., that a clique in F can be used to generate α.

Lemma 11. *Let α be an AMO such that every topological ordering that represents α has the minimal separator S as prefix. Then every $F \in \mathcal{B}(S)$ contains a clique K such that there is a $\tau \in \mathrm{top}(\alpha)$ starting with K.*

We use $\prec_T$ to define, for a fixed AMO α, a partial order $\prec_\alpha$ on the set of maximal cliques, which are at the beginning of some $\tau \in \mathrm{top}(\alpha)$, as follows: $K_1 \prec_\alpha K_2$ if, and only if, (i) $K_1 \cap K_2 = S \in \Delta(G)$, (ii) K_1 and K_2 are in S-flowers $F_1, F_2 \in \mathcal{B}(S)$, respectively, and (iii) $F_1 \prec_T F_2$.

Proposition 3. *Let G be a UCCG and $\mathcal{T} = (T,r,\iota)$ be a rooted clique tree of G. Then:*

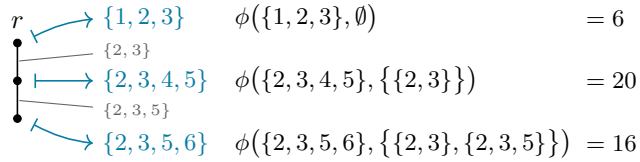
$$\#\mathrm{AMO}(G) = \sum_{v \in V_T} \phi(\iota(v), \mathrm{FP}(v,\mathcal{T})) \times \prod_{H \in \mathcal{C}_G(\iota(v))} \#\mathrm{AMO}(H).$$

Sketch of Proof. First, prove that for every AMO α there is a unique least $K \in \Pi(G)$ with respect to $\prec_\alpha$. Let x be the node of the clique tree with $\iota(x) = K$, we deduce that α is counted in $\phi(\iota(x), \mathrm{FP}(x,\mathcal{T}))$, but is blocked in all other nodes y by some set in $\mathrm{FP}(y,\mathcal{T})$. $\square$

Algorithm 2 evaluates this formula, utilizing memoization to avoid recomputations. Traversing the clique tree with a BFS allows for simple computation of FP.

Theorem 2. *For an input UCCG G, Algorithm 2 returns the number of AMOs of G.*

Example 5. We consider a rooted clique tree (T,r,ι) for the graph G from Example 3. The root is labeled with r and the function ι is visualized in blue. The edges of the clique tree are labeled with the corresponding minimal separators.



Algorithm 2 traverses the tree T from the root r to the bottom and computes the values shown at the right. The only case in which we obtain a non-trivial subgraph is for $S = \{1,2,3\}$ (an induced path on $\{4,5,6\}$). Therefore:

$$\#\mathrm{AMO}(G) = 6 \cdot 3 + 20 \cdot 1 + 16 \cdot 1 = 54. \qquad \diamond$$

input : A UCCG $G = (V,E)$.
output: #AMO(G).

```
function count(G, memo)
    if G ∈ memo then
        return memo[G]
    end
    T = (T, r, ι) ← a rooted clique tree of G
    sum ← 0
    Q ← queue with single element r
    while Q is not empty do
        v ← pop(Q)
        push(Q, children(v))
        prod ← 1
        foreach H ∈ C_G(ι(v)) do
            prod ← prod · count(H, memo)
        end
        sum ← sum + φ(ι(v), FP(v, T)) · prod
    end
    memo[G] = sum
    return sum
end
```

Algorithm 2: The Clique-Picking algorithm computes the number of acyclic moral orientations of a UCCG G.

Since clique trees can be computed in linear time (Blair and Peyton 1993), an iteration of the algorithm runs in polynomial time due to Lemma 6 and 7. We prove in the next section that Algorithm 2 performs at most $2 \cdot |\Pi(G)| - 1$ recursive calls, which implies overall polynomial run time.

6 The Complexity of #AMO

We analyze the run time of the Clique-Picking algorithm by bounding the number of connected chordal subgraphs that we encounter. The following proposition shows that this number can be bounded by $\mathcal{O}(|\Pi(G)|)$. Recall that we have $|\Pi(G)| \leq |V|$ in chordal graphs and, thus, we only have to handle a linear number of recursive calls.

Proposition 4. *Let G be a UCCG. The number of distinct UCCGs explored by* `count` *is bounded by $2|\Pi(G)| - 1$.*

Sketch of Proof. We observe that there is a bijection between S-flowers of G and distinct UCCGs. The linear bound follows, as a separator S, that has a bouquet of size k, is associated with $k-1$ edges of the clique tree. Therefore, we have at most $|\Pi(G)| - 1 - (k-1)$ further separators and the maximum number of flowers is obtained if the quotient $k/(k-1)$ is maximized – which is the case for $k = 2$. $\square$

We are now able to bound the run time of Clique-Picking:

Theorem 3. *The Clique-Picking algorithm runs in time $\mathcal{O}\big(|\Pi(G)|^2 \cdot (|V| + |E|)\big)$.*

Sketch of Proof. The algorithm explores $\mathcal{O}(|\Pi(G)|)$ distinct UCCGs by Proposition 4. For each UCCG we compute a clique tree, and for all nodes $v \in V_T$ the set $\mathcal{C}_G(\iota(v))$ and the value $\phi(\iota(v), \mathrm{FP}(v,\mathcal{T}))$. Both can be done in time $\mathcal{O}(|V| + |E|)$ by Theorem 1 and by using the formula from Lemma 6. $\square$

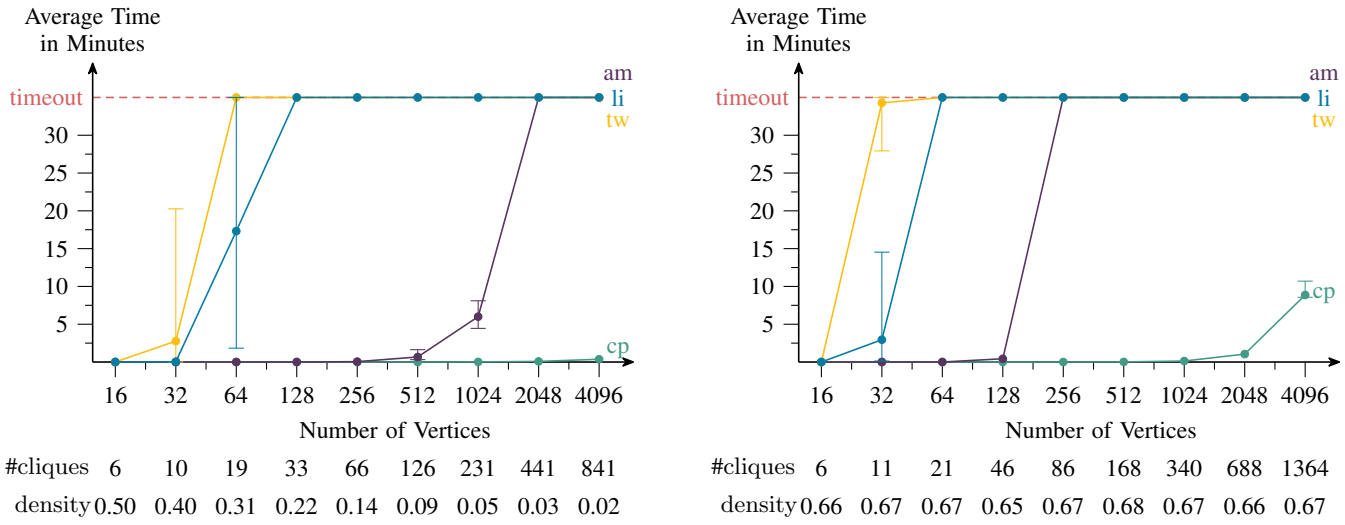


Figure 2: Experimental results for the solvers Clique-Picking (cp), AnonMAO (am), TreeMAO (tw), and LazyIter (li) on random chordal graphs with $n = 16, 32, \ldots, 4096$ vertices. For the left plot, we used graphs generated with the subtree intersection method and density parameter $k = \log n$; the right plot contains the results for random interval graphs. At the bottom, we present the number of maximal cliques as well as the graph density $|E|/\binom{|V|}{2}$.

As one would expect, the Clique-Picking algorithm can – with slight modifications – also be used to sample Markov equivalent DAGs uniformly at random. Hence, this problem can be solved in polynomial time, too.

Theorem 4. *There is an algorithm that, given a connected chordal graph G, uniformly samples AMOs of G in time $\mathcal{O}(|V| + |E|)$ after an initial $\mathcal{O}(\Pi(G)^2 \cdot |V| \cdot |E|)$ setup.*

Sketch of Proof. We sample the AMOs recursively: Draw a clique K proportional to the number of AMOs counted at this clique; uniformly draw a permutation of K that does not start with a forbidden prefix; recurse on subgraphs. □

We summarize the findings of this section:[3]

Theorem 5. *The problems* #AMO *and uniform sampling from a Markov equivalence class are in* P.

The following theorem shows that Theorem 5 is tight in the sense that counting Markov equivalent DAGs that encode additional background knowledge (e. g., that are represented as so-called *PDAGs* or *MPDAGs*) is not in P under standard complexity-theoretic assumptions.

Theorem 6. *The problem of counting the number of AMOs is* #P*-complete for PDAGs and MPDAGs.*

7 Experimental Evaluation of Clique-Picking

We evaluate the practical performance of the Clique-Picking algorithm by comparing it to three state-of-the-art algorithms for #AMO. *AnonMAO* (Ganian, Hamm, and Talvitie 2020) is the best root-picking method; *TreeMAO* (Talvitie and Koivisto 2019) utilizes dynamic programming on the clique tree; and *LazyIter* (AhmadiTeshnizi, Salehkaleybar, and Kiyavash 2020) combines techniques from intervention design with dynamic programming.

Figure 2 shows the run time of the four algorithms on random chordal graphs – details of the random graph generation and further experiments can be found in the supplementary material[4]. We chose the random subtree intersection method (left plot in Fig. 2) as it generates a broad range of chordal graphs (Seker et al. 2017); and we complemented these with random interval graphs (right plot) as *AnonMAO* runs provably in polynomial time on this subclass of chordal graphs (Ganian, Hamm, and Talvitie 2020).

The Clique-Picking algorithm outperforms its competitors in both settings. For the subtree intersection graphs, it solves all instances in less than a minute, while the other solvers are not able to solve instances with more than 1024 vertices. The large instances of the interval graphs are more challenging, as they are denser and have more maximal cliques. However, Clique-Picking is still able to solve all instances, while the best competitor, *AnonMAO*, can not handle graphs with 256 or more vertices.

8 Conclusion

We presented the first polynomial-time algorithms for counting and sampling Markov equivalent DAGs. Our novel Clique-Picking approach utilizes the clique tree without applying cumbersome dynamic programming on it. As a result, the algorithm is not only of theoretical but also of high practical value, being the fastest algorithm by a large margin.

[3] Additionally, we remark that the size of interventional Markov equivalence classes can also be computed in polynomial time.

[4] The source code and supplementary material are available at https://github.com/mwien/CliquePicking.

Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) grant LI634/4-2.

The authors thank Paula Arnold for her help in setting up the experiments.

References

AhmadiTeshnizi, A.; Salehkaleybar, S.; and Kiyavash, N. 2020. LazyIter: A Fast Algorithm for Counting Markov Equivalent DAGs and Designing Experiments. In *Proceedings of the 37th International Conference on Machine Learning, ICML '20*.

Andersson, S. A.; Madigan, D.; and Perlman, M. D. 1997. A Characterization of Markov Equivalence Classes for Acyclic Digraphs. *The Annals of Statistics* 25(2): 505–541.

Blair, J. R.; and Peyton, B. 1993. An Introduction to Chordal Graphs and Clique Trees. In *Graph Theory and Sparse Matrix Computation*, 1–29. Springer.

Brightwell, G. R.; and Winkler, P. 1991. Counting Linear Extensions is #P-Complete. In *Proceedings of the 23th Annual ACM Symposium on Theory of Computing, STOC '91*, 175–181.

Chickering, D. M. 2002a. Learning Equivalence Classes of Bayesian-Network Structures. *Journal of Machine Learning Research* 2: 445–498.

Chickering, D. M. 2002b. Optimal Structure Identification With Greedy Search. *Journal of Machine Learning Research* 3: 507–554.

Dirac, G. A. 1961. On Rigid Circuit Graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 25(1): 71–76.

Fulkerson, D.; and Gross, O. 1965. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics* 15(3): 835–855.

Ganian, R.; Hamm, T.; and Talvitie, T. 2020. An Efficient Algorithm for Counting Markov Equivalent DAGs. In *Proccedings of the 34th Conference on Artificial Intelligence, AAAI 20*, 10136–10143.

Ghassami, A.; Salehkaleybar, S.; Kiyavash, N.; and Bareinboim, E. 2018. Budgeted Experiment Design for Causal Structure Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML '18*, 1719–1728.

Ghassami, A.; Salehkaleybar, S.; Kiyavash, N.; and Zhang, K. 2019. Counting and Sampling from Markov Equivalent DAGs Using Clique Trees. In *Proccedings of the 33th Conference on Artificial Intelligence, AAAI 19*, 3664–3671.

Gillispie, S. B.; and Perlman, M. D. 2002. The Size Distribution for Markov Equivalence Classes of Acyclic Digraph Models. *Artificial Intelligence* 141(1/2): 137–155.

Hauser, A.; and Bühlmann, P. 2012. Characterization and Greedy Learning of Interventional Markov Equivalence Classes of Directed Acyclic Graphs. *Journal of Machine Learning Research* 13: 2409–2464.

He, Y.; Jia, J.; and Yu, B. 2015. Counting and Exploring Sizes of Markov Equivalence Classes of Directed Acyclic Graphs. *Journal of Machine Learning Research* 16(79): 2589–2609.

He, Y.-B.; and Geng, Z. 2008. Active Learning of Causal Networks with Intervention Experiments and Optimal Designs. *Journal of Machine Learning Research* 9(Nov): 2523–2547.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning* 20(3): 197–243.

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

Maathuis, M. H.; Kalisch, M.; and Bühlmann, P. 2009. Estimating High-Dimensional Intervention Effects from Observational Data. *The Annals of Statistics* 37(6A): 3133–3164.

Madigan, D.; Andersson, S. A.; Perlman, M. D.; and Volinsky, C. T. 1996. Bayesian Model Averaging and Model Selection for Markov Equivalence Classes of Acyclic Digraphs. *Communications in Statistics–Theory and Methods* 25(11): 2493–2519.

Meek, C. 1995. Causal Inference and Causal Explanation with Background Knowledge. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, UAI '95*, 403–410.

Meek, C. 1997. *Graphical Models: Selecting Causal and Statistical Models*. Ph.D. thesis, Carnegie Mellon University.

Pearl, J. 2009. *Causality*. Cambridge University Press. ISBN 978-0521895606.

Perkovic, E.; Textor, J.; Kalisch, M.; and Maathuis, M. H. 2017. Complete Graphical Characterization and Construction of Adjustment Sets in Markov Equivalence Classes of Ancestral Graphs. *Journal of Machine Learning Research* 18: 220:1–220:62.

Rodionov, A. S.; and Choo, H. 2004. On Generating Random Network Structures: Connected Graphs. In *Proccedings of the xth Internation Conference on Information Networking, ICOIN 04*, volume 3090, 483–491.

Rose, D. J.; Tarjan, R. E.; and Lueker, G. S. 1976. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing* 5(2): 266–283.

Scheinerman, E. R. 1988. Random Interval Graphs. *Combinatorica* 8(4): 357–371.

Seker, O.; Heggernes, P.; Ekim, T.; and Taskin, Z. C. 2017. Linear-Time Generation of Random Chordal Graphs. In *Proccedings of the 10th International Conference on Algorithms and Complexity, CIAC 17*, volume 10236, 442–453.

Shanmugam, K.; Kocaoglu, M.; Dimakis, A. G.; and Vishwanath, S. 2015. Learning Causal Graphs with Small Interventions. In *Processing of the 28th Annual Conference on Neural Information Processing Systems, NIPS '15*, 3195–3203.

Spirtes, P.; Glymour, C.; and Scheines, R. 2000. *Causation, Prediction, and Search, Second Edition*. MIT Press. ISBN 978-0-262-19440-2.

Talvitie, T.; and Koivisto, M. 2019. Counting and Sampling Markov Equivalent Directed Acyclic Graphs. In *Proccedings of the 33th Conference on Artificial Intelligence, AAAI 19*, 7984–7991.

van der Zander, B.; and Liśkiewicz, M. 2016. Separators and Adjustment Sets in Markov Equivalent DAGs. In *Proceedings of the 30th Conference on Artificial Intelligence, AAAI '16*, 3315–3321.

Verma, T.; and Pearl, J. 1990. Equivalence and Synthesis of Causal Models. In *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence, UAI '90*, 255–270.

Verma, T.; and Pearl, J. 1992. An Algorithm for Deciding if a Set of Observed Independencies has a Causal Explanation. In *Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence, UAI '92*, 323–330.

Vose, M. D. 1991. A Linear Algorithm For Generating Random Numbers With a Given Distribution. *IEEE Transactions on Software Engineering* 17(9): 972–975.

Appendix

Part I
Detailed Proofs

We present the detailed proofs that are missing within the main paper. This part of the appendix is structured as the main paper, i. e., for every section of the paper (that contains lemmas or theorems) there is a section here that contains the corresponding detailed proofs. For the reader's convenience, we repeated the statements of the lemmas and theorems and, if appropriate, recall some central definitions. Some of the proofs require additional auxiliary lemmas that did not appear within the main text. These new lemmas are marked with a $\star$.

Proofs of Section 3: Lexicographic BFS and AMOs

Claim of Lemma 1. *A topological ordering τ of the vertices of a UCCG G represents an AMO if, and only if, it is the reverse of a perfect elimination ordering.*

Proof. For the first direction, assume τ is a topological ordering representing an AMO. By definition of AMOs, there can not be a v-structure and, thus, if two vertices $x, y \in N(u)$ precede u in τ, they need to be neighbors. This implies that the neighbors of u preceding u in τ form a clique. Thus, the reverse of τ is a perfect elimination ordering.

For the second direction, assume ρ is a perfect elimination ordering and orient the edges according to the topological ordering that is the reverse of ρ. Clearly, the orientation is acyclic. Moreover, there can be no v-structure, as two vertices x, y preceding u in the reverse of ρ are neighbors. Thus, the reverse of ρ represents an AMO. □

Claim of Corollary 1. *Every LBFS ordering τ of a UCCG G represents an AMO.*

Proof. Follows immediately from Lemma 1 and the fact that an LBFS always outputs a PEO when performed on a chordal graph (Rose, Tarjan, and Lueker 1976). □

In order to prove the next lemmas of the main paper, we require further auxiliary lemmas and definitions. A *state* $\mathcal{X}$ of the LBFS algorithm is a tuple $(V(\mathcal{X}), \mathcal{S}(\mathcal{X}))$, where $V(\mathcal{X})$ is the sequence of already visited vertices at some point of the algorithm and $\mathcal{S}(\mathcal{X})$ the current sequence of sets after the last vertex in $V(\mathcal{X})$ has been processed. From now on, the term *preceding neighbors* of v at state $\mathcal{X}$ (denoted by $P_{\mathcal{X}}(v)$) describes the set $N(v) \cap V(\mathcal{X})$, i. e., all neighbors of v which were visited before v at state $\mathcal{X}$ of the LBFS.

$\star$ **Lemma 12.** *In a state $\mathcal{X}$ of the LBFS algorithm performed on a chordal graph G, two unvisited vertices u and v are in the same set if, and only if, the preceding neighbors of u and v are identical, i. e., $P_{\mathcal{X}}(u) = P_{\mathcal{X}}(v)$. These preceding neighbors form a clique.*

Proof. The second statement follows from the fact that the LBFS algorithm produces a valid PEO for chordal graphs. Thus, when a vertex w is visited, all preceding neighbors form a clique. This also holds for subsets of these neighbors and, hence, for every state $\mathcal{X}$.

Let us now prove the first statement. First, observe that $P_{\mathcal{X}}(u) = P_{\mathcal{X}}(v)$ implies that u and v are in the same set. For each visited vertex, the sets are partitioned as described in line 13 of Algorithm 1. As this visited vertex is either a neighbor of u and v or of neither of them, these vertices are put in the same set.

We show the other direction by contradiction. Assume u and v are in the same set, but the sets of preceding neighbors differ, i. e., $P_{\mathcal{X}}(u) \neq P_{\mathcal{X}}(v)$. Moreover, assume without loss of generality that x is the earliest visited vertex that is (i) in exactly one of these sets and that (ii) is a neighbor of u but not v. When x is visited, u and v are in the same set by the argument above (the sets of preceding neighbors of u and v are identical in this state) and, according to the partitioning in line 13, u and v are put in different sets in $\mathcal{S}$. As no sets are joined in the LBFS, it follows that u and v stay in different sets and, thus, are not in the same set. A contradiction. □

Due to this result, we will also refer to the *preceding neighbors of a set $S \in \mathcal{S}(\mathcal{X})$* at state $\mathcal{X}$ as $P_{\mathcal{X}}(S)$, which means the preceding neighbors of any node in S.

$\star$ **Lemma 13.** *Let $\mathcal{X}$ be the state of an LBFS performed on a chordal graph $G = (V, E)$, and let $S_1, S_2 \in \mathcal{S}(\mathcal{X})$ such that S_1 precedes S_2 in $\mathcal{S}(\mathcal{X})$ and such that there are vertices $u \in S_1$ and $v \in S_2$ with $\{u, v\} \in E$. Then $P_{\mathcal{X}}(S_1) \supsetneq P_{\mathcal{X}}(S_2)$.*

Proof. Assume for the sake of contradiction that there is a vertex $x \in P_{\mathcal{X}}(S_2) \setminus P_{\mathcal{X}}(S_1)$. Let $u \in S_1$ and $v \in S_2$ be the vertices with edge $u - v$.

Vertex v is visited after u by the LBFS, as S_2 succeeds S_1. Since x was already visited as well, we have that the preceding neighbors of v (when v is visited) do not form a clique. This contradicts the fact that the LBFS ordering is the reverse of a PEO. □

Claim of Lemma 2. *Every AMO of a UCCG G can be represented by an LBFS ordering.*

Proof. We show how the LBFS algorithm can be used to obtain a topological ordering that represents an AMO α.

The LBFS algorithm can freely choose a vertex from the first set. We restrict the algorithm to choose a vertex that has no incoming edges in α from an unvisited vertex. If such a vertex always exists, we obtain a topological ordering which represents α.

It is left to show that such a vertex indeed exists. We prove this by induction, where the base case is the source vertex of α, which can be chosen, as all vertices are in the first set. Assume the first $k - 1$ vertices were chosen from the (at that state) first set. We show that it is possible to pick a kth

vertex from the first set, which has no incoming edges from an unvisited vertex.

Let $\mathcal{X} = (V(\mathcal{X}), \mathcal{S}(\mathcal{X}))$ be the state when picking the kth vertex and assume, for the sake of contradiction, that there is no such vertex in the first set of $\mathcal{S}(\mathcal{X})$. This means, for all vertices x in the first set X of $\mathcal{S}(\mathcal{X})$, an unvisited vertex z exists with $x \leftarrow z$ in α. There has to be at least one z that is not in X, as otherwise there would be a cycle in α. This z is in a later set than x, and there is an edge between x and z. We deduce $P_{\mathcal{X}}(z) \subsetneq P_{\mathcal{X}}(x)$ with Lemma 13.

Let $u \in P_{\mathcal{X}}(x) \setminus P_{\mathcal{X}}(z)$ be a preceding neighbor of x but not z. The edge between u and v is correctly directed $u \rightarrow x$ by induction hypothesis. However, if α would contain the edge $x \leftarrow z$, then there would be a v-structure in α, as there is no edge between u and z. A contradiction. □

Claim of Lemma 3. *Every LBFS ordering starts with a maximal clique.*

Proof. At the beginning of the LBFS, an arbitrary vertex u is chosen. In the subsequent steps, as long as there exists a vertex that is adjacent to all previously visited vertices, all vertices in the first set in $\mathcal{S}(\mathcal{X})$ have this property (in line 13 of Algorithm 1 they are put in front sets of previously visited vertices). □

Claim of Corollary 2. *Every AMO can be represented by a topological ordering which starts with a maximal clique.*

Proof. By Lemma 3, every LBFS ordering starts with a maximal clique. The statement follows from the fact that every AMO can be represented by an LBFS ordering (Lemma 2). □

For the sake of readability, we repeat the following definition from the main paper:

Definition 1. *Let G be a UCCG over V, K be a clique in G, and let $\pi(K)$ be a permutation of K.*

1. *The $\pi(K)$-orientation of G, also denoted $G^{\pi(K)}$, is the union of all AMOs of G that can be represented by a topological ordering beginning with $\pi(K)$.*
2. *Let G^K denote the union of $\pi(K)$-orientations of G over all π, i. e., let $G^K = \bigcup_\pi G^{\pi(K)}$.*
3. *Denote by $\mathcal{C}_G(\pi(K))$ the undirected connected components of $G^{\pi(K)}[V \setminus K]$ and let $\mathcal{C}_G(K)$ denote the undirected connected components of $G^K[V \setminus K]$.*

We introduce a linear-time algorithm for finding the undirected components of $G^{\pi(K)}$. Algorithm 3 proceeds as Algorithm 1 in the main paper, with the exception that it makes sure that the vertices of the clique K are visited in order $\pi(K)$. In this way, the algorithm characterizes exactly the $\pi(K)$-orientation, which makes the proofs cleaner compared to showing the correctness of Algorithm 1 directly. Afterward, in Proposition 1, we observe that indeed the permutation $\pi(K)$ does not influence orientations beyond the initial clique. This will allow us to immediately conclude the correctness of Algorithm 1 from the main paper (Theorem 1).

input : A UCCG G, clique K, and permutation $\pi(K)$.
output: $\mathcal{C}_G(\pi(K))$.

```
1  S ← sequence of sets initialized with (K, V \ K)
2  τ ← empty list, L ← ∅
3  while S is non-empty do
4  |  X ← first non-empty set of S
5  |  if X ⊆ K then
6  |  |  v ← the first vertex of X in π(K)
7  |  else
8  |  |  v ← arbitrary vertex from X
9  |  end
10 |  Add vertex v to the end of τ.
11 |  if v is neither in a set in L nor in K then
12 |  |  L ← L ∪ {X}
13 |  |  Output the undirected components of G[X].
14 |  end
15 |  X ← X \ {v}
16 |  Denote the current S by (S_1, ..., S_k).
17 |  Replace each S_i by S_i ∩ N(v), S_i \ N(v).
18 |  Remove all empty sets from S.
19 end
```

Algorithm 3: The algorithm computes $\mathcal{C}_G(\pi(K))$.

⋆ **Lemma 14.** *Algorithm 3 computes the undirected connected components of $G^{\pi(K)}$ in time $\mathcal{O}(|V| + |E|)$. Moreover, all remaining directed edges in $G^{\pi(K)}$ are oriented as given by LBFS ordering τ.*

Proof. We first show that the edges between vertices in any set in L are correctly left undirected, i. e., they are undirected in $G^{\pi(K)}$. Note that by definition of $G^{\pi(K)}$ an edge $u - v$ is undirected, if there are AMOs represented by topological orderings that have $\pi(K)$ at the beginning and orient $u \rightarrow v$ and $u \leftarrow v$, respectively.

Note that the algorithm starts with vertices in K in the order $\pi(K)$. By Corollary 1, the LBFS ordering will always represent an AMO with these properties.

Whenever, in a certain state $\mathcal{X}$ (after the initial clique K has been visited), the algorithm chooses from the first set $X \in \mathcal{S}(\mathcal{X})$, it can choose the vertex arbitrarily. For any two neighbors u and v in X, there is an AMO with $u \rightarrow v$ (if we choose u as first vertex) and one with $u \leftarrow v$ (if we choose v). Thus, the edge between u and v is undirected in $G^{\pi(K)}$. Hence, when a set X is appended to L in line 12 of Algorithm 3, edges in $G^{\pi(K)}[X]$ are undirected.

We show that all remaining edges are oriented in the way given by the LBFS ordering τ in $G^{\pi(K)}$. Clearly, the internal edges of K are oriented correctly. We prove the correctness of the remaining edges by induction: For each vertex u, we show that the edge to every vertex v, which comes after it in the LBFS and is not in the same set in L, is directed as $u \rightarrow v$ in $G^{\pi(K)}$. This means that every AMO of G whose topological ordering starts with $\pi(K)$ contains the edge $u \rightarrow v$.

This holds for K, as edges to vertices that are not in K are always directed outwards from K. Assume the stated

property holds for all previous vertices in the LBFS ordering τ, we show it also holds for vertex v. Let $\mathcal{X}$ be the state of the LBFS before the first vertex from the set of v in L was visited. For any subsequent vertex w, which is not in the same set in L as v, there is a preceding neighbor u of v at state $\mathcal{X}$, which is not a preceding neighbor of w at state $\mathcal{X}$. Otherwise w would have been in the same set at state $\mathcal{X}$ by Lemma 12 and, thereby, be in the same set in L as v. But then we have $u \to v - w$, with the correctness of edge $u \to v$ following from the induction hypothesis (u cannot be in the same set in L as v by definition of $\mathcal{X}$). By the first Meek rule, it follows that $v \to w$ is in every AMO of $G^{\pi(K)}$.

The sets in L do not necessarily induce connected subgraphs. Thus, the algorithm returns the connected components of these sets, which are exactly the undirected connected components of $G^{\pi(k)}$.

For the run time observe that the algorithm can be implemented in linear time with the same techniques used to implement the standard LBFS. □

We need further vocabulary to prove the next lemma. Let $H \in \mathcal{C}_G(\pi(K))$ be returned by Algorithm 3. We denote the set of preceding neighbors of the vertices in H at state $\mathcal{X}_{\text{out}}$, the state when H was output, by $P_{\text{out}}(H)$. Note that this is a slight abuse of notation as H is not a set in $\mathcal{S}$ but a graph.

Claim of Lemma 4. *The undirected components in $\mathcal{C}_G(\pi(K))$ are chordal and it holds that:*

$$\#\text{AMO}(G^{\pi(K)}) = \prod_{H \in \mathcal{C}_G(\pi(K))} \#\text{AMO}(H).$$

Proof. The chordality of the graphs in $\mathcal{C}_G(\pi(K))$ follows from the correctness of Algorithm 3 (Lemma 14) and the fact that the undirected components returned by Algorithm 3 are induced subgraphs of G.

The second part, i. e., the recursive formula, follows from the property that each component H in $\mathcal{C}_G(\pi(K))$ can be oriented independently of the directed part of $G^{\pi(K)}$. To prove this, we have to show that whenever a vertex u is a parent of vertex $v \in V_H$, then u is a parent of all vertices in H. This fact was proven for CPDAGs as Lemma 10 by He and Geng (2008) and the property then follows analogously for the UCCGs of the $\pi(K)$-orientation as in Theorem 4 and Theorem 5 of the same paper.

By Lemma 14, the parents of a vertex $v \in V_H$ in the $\pi(K)$-orientation are the vertices in $P_{\text{out}}(H)$. All vertices in this set are neighbors of v, considered before v and not in the same set in L. All other neighbors of v are visited later or are in the same set in L as v and can, thus, not be parents. The set $P_{\text{out}}(H)$ is by definition the same for each vertex in H. □

Claim of Proposition 1. *Let G be a UCCG and K be a clique of G. For each permutation $\pi(K)$ it is true that all edges of $G^{\pi(K)}$ coincide with the edges of G^K, excluding the edges connecting the vertices in K. Hence, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(K)$ and it holds that:*

$$\sum_{\pi \text{ over } K} \#\text{AMO}(G^{\pi(K)}) = |K|! \times \prod_{H \in \mathcal{C}_G(K)} \#\text{AMO}(H).$$

Proof. We prove the statement by showing that, for two arbitrary permutations $\pi(K)$ and $\pi'(K)$, the edges in $G^{\pi(K)}$ and $G^{\pi'(K)}$ coincide, excluding the edges connecting the vertices in K.

The graph $G^{\pi(K)}$ is defined as the union of all AMOs, which can be represented by a topological ordering starting with $\pi(K)$. Take such an AMO α and, in a corresponding topological ordering τ, replace $\pi(K)$ by $\pi'(K)$ obtaining a new topological ordering τ'. The orientation α' represented by τ' is, by definition, acyclic and, moreover, moral. For the latter property, assume for a contradiction, that there is a v-structure (immorality) $a \to b \leftarrow c$. Because α is moral and only edge directions internal in K have been changed in α', it has to hold that either

1. two vertices of a, b, c are in K (w.l.o.g. assume these are a and b), but then we have $b \to c \notin K$ and not $b \leftarrow c$, or
2. all three vertices are in K, but then $a \to b \leftarrow c$ is no induced subgraph.

Hence, such a v-structure can not exist and α' is moral as well. The reverse direction follows equivalently.

Therefore, the union of all AMOs, which can be represented by a topological ordering τ' starting with $\pi'(K)$, yields the exact same graph as for $G^{\pi(K)}$, excluding the internal edges in K. Thus, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(\pi'(K))$ and, by definition, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(K)$. The recursive formula is immediately implied by this fact and Lemma 4. □

Claim of Theorem 1. *For a chordal graph G and a clique K, Algorithm 1 computes $\mathcal{C}_G(K)$ in time $\mathcal{O}(|V| + |E|)$.*

Proof. By Lemma 14, Algorithm 3 correctly computes the UCCGs of the $\pi(K)$-orientation. As these UCCGs are identical for each orientation $\pi(K)$ and Algorithm 1 proceeds just as Algorithm 3, with the only difference that the former makes no restriction on the order the vertices in K are visited (thus visiting them in arbitrary permutation $\pi'(K)$), the correctness follows. □

Proofs of Section 4: Counting MA-Orientations with Minimal Separators and Maximal Cliques

Claim of Lemma 5. *Let α be an AMO of a chordal graph G and τ_1, τ_2 be two topological orderings that represent α. Then τ_1 and τ_2 have a common prefix $S \in \Delta(G) \cup \Pi(G)$.*

Proof. Assume by Corollary 2 that τ_1 starts with the maximal clique K_1 and τ_2 with the maximal clique K_2. Since every AMO of a UCCG has a unique source, τ_1 and τ_2 start with the same vertex and, hence, $K_1 \cap K_2 = S \neq \emptyset$.

We first show that τ_1 and τ_2 have to start with S. Assume for a contradiction that in τ_1 there is a vertex $u \notin S$ before a $v \in S$. The edge between u and v is directed as $u \to v$ in α, but as $v \in K_2$ and $u \notin K_2$, the ordering τ_2 implies $u \leftarrow v$.

If $K_1 = K_2$ then $S \in \Pi(G)$ and we are done. We prove that otherwise S is a minimal separator in G that separates $P_1 = K_1 \setminus S$ from $P_2 = K_2 \setminus S$. Note that the minimality follows by definition. It remains to show

that S indeed separates P_1 and P_2. For a contradiction, let $x_1 \in P_1 - x_2 - \cdots - x_{k-1} - x_k \in P_2$ be a shortest P_1-P_2-path in $G[V \setminus S]$ with $x_i \notin K_1 \cup K_2$ for $i \in \{2, \ldots, k-1\}$. According to τ_1, we have the edge $x_1 \to x_2$ in α. Since we consider a shortest path, $x_{i-1} - x_i - x_{i+1}$ is always an induced subgraph and, thus, an iterative application of the first Meek rule implies $x_{k-1} \to x_k$. However, τ_2 would imply the edge $x_{k-1} \leftarrow x_k$ in α. A contradiction. □

We repeat the following definition from the main text:

Definition 2. *For a set S and a collection R of subsets of S, we define $\phi(S, R)$ as the number of all permutations of S that do not have a set $S' \in R$ as prefix.*

Claim of Proposition 2. *Let G be a UCCG. Then:*

$$\#\mathrm{AMO}(G) = \sum_{S \in \Delta(G) \cup \Pi(G)} \phi(S) \times \prod_{H \,\in\, \mathcal{C}_G(S)} \#\mathrm{AMO}(H).$$

Proof. By the choice of S and the definition of $\mathcal{C}_G(S)$, everything counted by the formula is a topological ordering representing an AMO. We argue that every AMO α is counted exactly once. Let $S \in \Delta(G) \cup \Pi(G)$ be the smallest common prefix of all topological orderings in $\mathrm{top}(\alpha)$ – which is well-defined by Lemma 5. First observe that, by the minimality of S, α is counted at the term for S: There is no other prefix $\tilde{S} \subsetneq S$ of the topological orderings with $\tilde{S} \in \Delta(G)$ and $\tilde{S} \in \phi(S)$.

On the other hand, S is the only term in the sum at which we can count α, as for any larger $\tilde{S}$ with $S \subsetneq \tilde{S}$ that is a prefix of some $\tau \in \mathrm{top}(\alpha)$, we have S is considered in $\phi(\tilde{S})$. □

Proofs of Section 5: The Clique-Picking Algorithm

Claim of Lemma 6. *Let S be a set and $R = \{X_1, \ldots, X_\ell\}$ be a collection of subsets of S with $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$. Then:*

$$\phi(S, R) = |S|! - \sum_{i=1}^{\ell} |S \setminus X_i|! \cdot \phi(X_i, \{X_1, \ldots, X_{i-1}\}).$$

Proof. We prove the statement by induction over ℓ with the base case $\phi(S, \emptyset) = |S|!$. Consider a set S and a collection $R = \{X_1, \ldots, X_\ell\}$ of subsets of S. We can compute $\phi(S, R)$ by taking $\phi(S, \{X_1, \ldots, X_{\ell-1}\})$ (the number of permutations of S that do not start with $X_1, \ldots, X_{\ell-1}$) and by subtracting the number of permutations that start with X_ℓ but none of the other X_i, i. e.,

$$\begin{aligned}\phi(S, R) = \phi(S, \{X_1, \ldots, X_{\ell-1}\}) \\ - |S \setminus X_\ell|! \cdot \phi(X_\ell, \{X_1, \ldots, X_{\ell-1}\}).\end{aligned}$$

Inserting the induction hypothesis, we obtain:

$$\begin{aligned}\phi(S, R) &= |S|! - \sum_{i=1}^{\ell-1} |S \setminus X_i|! \cdot \phi(X_i, \{X_1, \ldots, X_{i-1}\}) \\ &\quad - |S \setminus X_\ell|! \cdot \phi(X_\ell, \{X_1, \ldots, X_{\ell-1}\}) \\ &= |S|! - \sum_{i=1}^{\ell} |S \setminus X_i|! \cdot \phi(X_i, \{X_1, \ldots, X_{i-1}\}).\end{aligned}$$

□

Recall, for the following proof, the definition of a forbidden prefix:

Definition 3. *Let G be a UCCG, $\mathcal{T} = (T, r, \iota)$ a rooted clique tree of G, v a node in T and $r = x_1, x_2, \ldots, x_p = v$ the unique r-v-path. We define the set $\mathrm{FP}(v, \mathcal{T})$ to contain all sets $\iota(x_i) \cap \iota(x_{i+1}) \subseteq \iota(v)$ for $1 \leq i < p$.*

Claim of Lemma 7. *We can order the elements of the set $\mathrm{FP}(v, \mathcal{T})$ as $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$.*

Proof. The ordering of the sets is given by the natural order along the path from the root r to node v. The sets in $\mathrm{FP}(v, \mathcal{T})$ satisfy $\iota(x_i) \cap \iota(x_{i+1}) \subseteq \iota(v)$. By the definition of a clique tree, we have $\iota(x_i) \cap \iota(x_{i+1}) \subseteq \iota(y)$ for each y that lies on the x_i-v-path in T. Hence, each such y can only add supersets of $\iota(x_i) \cap \iota(x_{i+1})$ to $\mathrm{FP}(v, \mathcal{T})$. □

Recall the definition of S-flowers and bouquets:

Definition 4. *An S-*flower *for a minimal separator S is a maximal set $F \subseteq \{ K \mid K \in \Pi(G) \wedge S \subseteq K \}$ such that $\bigcup_{K \in F} K$ is connected in $G[V \setminus S]$. The* bouquet *$\mathcal{B}(S)$ of a minimal separator S is the set of all S-flowers.*

Claim of Lemma 8. *An S-flower F is a connected subtree in a rooted clique tree (T, r, ι).*

Proof. Assume for a contradiction that F is not connected in T. Then there are cliques $K_1, K_2 \in F$ that are connected by the unique path $K_1 - \tilde{K} - \cdots - K_2$ with $\tilde{K} \notin F$. Since $\iota^{-1}(S)$ is connected, we have $S \subseteq \tilde{K}$. By the maximality of F, we have $K_1 \cap \tilde{K} = S$. But then S separates $K_1 \setminus S$ from $K_2 \setminus S$, which contradicts the definition of S-flowers. □

Claim of Lemma 9. *For any minimal separator S, the bouquet $\mathcal{B}(S)$ is a partition of $\iota^{-1}(S)$.*

Proof. For each $x \in \iota^{-1}(S)$, the maximal clique $\iota(x)$ is in some S-flower by definition. However, no maximal clique can be in two S-flowers, as these flowers would then be in the same connected component in $G[V \setminus S]$. □

Claim of Lemma 10. *There is a unique least S-flower in $\mathcal{B}(S)$ with respect to $\prec_T$.*

Proof. Assume, there is no unique least S-flower. Then there are two minimal S-flowers which are incomparable. However, by Lemma 8 and 9, and the definition of the partial order, there has to be another S-flower closer to the root and, thus, lesser given the partial order. A contradiction. □

Claim of Lemma 11. *Let α be an AMO such that every topological ordering that represents α has the minimal separator S as prefix. Then every $F \in \mathcal{B}(S)$ contains a clique K such that there is a $\tau \in \mathrm{top}(\alpha)$ starting with K.*

Proof. Let τ be a topological ordering representing α that starts with S. By Proposition 2, there is at least one clique K with $S \subseteq K$ such that τ has the form $\tau = (S, K \setminus S, V \setminus K)$. Let $F \in \mathcal{B}(S)$ be the flower containing K and $F' \neq F$ be another S-flower with some $K' \in F'$. Observe that $K \setminus S$ is disconnected from $K' \setminus S$ in $G[V \setminus S]$. Therefore, there is

a topological ordering of the form $(S, K' \setminus S, V \setminus K')$ that represents α as well. □

Claim of Proposition 3. *Let G be a UCCG and $\mathcal{T} = (T, r, \iota)$ be a rooted clique tree of G. Then:*

$$\#\mathrm{AMO}(G) = \sum_{v \in V_T} \phi(\iota(v), \mathrm{FP}(v, \mathcal{T})) \times \prod_{H \in \mathcal{C}_G(\iota(v))} \#\mathrm{AMO}(H).$$

Proof. We have to show that every AMO α is counted exactly once. Recall that $\mathrm{top}(\alpha) = \{\tau_1, \ldots, \tau_\ell\}$ is the set of topological orderings that represent α, and that the rooted clique tree (T, r, ι) implies a partial order $\prec_T$ on flowers, which in return defines partial order $\prec_\alpha$ on the set of maximal cliques that are at the beginning of some $\tau \in \mathrm{top}(\alpha)$.

Claim 1. *There is a unique least maximal clique $K \in \Pi(G)$ with respect to $\prec_\alpha$.*

Proof. Let $\mathrm{top}'(\alpha) \subseteq \mathrm{top}(\alpha)$ be an arbitrary subset of the topological orderings that represent α and let μ be the number of different maximal cliques with which elements in $\mathrm{top}'(\alpha)$ start. We prove the claim by induction over μ. In the base case, all elements in $\mathrm{top}'(\alpha)$ start with the same set $S \in \Pi(G)$ and, of course, this is the unique least maximal clique. For $\mu > 1$ we observe that, by Lemma 5, all $\tau \in \mathrm{top}'(\alpha)$ start with the same $S \in \Delta(G)$.

Consider the bouquet $\mathcal{B}(S)$, which is partially ordered by $\prec_T$. Lemma 10 states that there is a unique least S-flower $F \in \mathcal{B}(S)$ with respect to $\prec_T$, and by the definition of $\prec_\alpha$ the maximal cliques occurring in F precede the others. Therefore, we reduce $\mathrm{top}'(\alpha)$ to the set $\mathrm{top}''(\alpha)$ of topological orderings that start with a maximal clique in F. This set is non-empty by Lemma 11 and contains, by the induction hypothesis, a unique least maximal clique. □

We complete the proof by showing that the formula counts α at the term for the unique least maximal clique K from the previous claim. To see this, we need to prove that (i) α can be counted at the clique K (i. e., there is no set $S \in \mathrm{FP}(\iota^{-1}(K), \mathcal{T})$ preventing α from being counted), and (ii) that α is not counted somewhere else (i. e., there is some $S \in \mathrm{FP}(\iota^{-1}(K'), \mathcal{T})$ for all other $K' \in \Pi(G)$ that can be at the beginning of some $\tau \in \mathrm{top}(\alpha)$).

Claim 2. *Let α be an AMO and $K \in \Pi(G)$ be the least maximal clique (with respect to $\prec_\alpha$) that is a prefix of some $\tau \in \mathrm{top}(\alpha)$. Then there is no $S \in \Delta(G)$ with $S \in \mathrm{FP}(\iota^{-1}(K), \mathcal{T})$ that is a prefix of τ.*

Proof. Assume for a contradiction that there would be such a $S \in \Delta(G)$ and let $F \in \mathcal{B}(S)$ be the S-flower containing K. Since $S \in \mathrm{FP}(\iota^{-1}(K), \mathcal{T})$, there is another flower $F' \in \mathcal{B}(S)$ with $F' \prec_T F$. Lemma 11 tells us that there is another clique $K' \in F'$ that is at the beginning of some $\tau' \in \mathrm{top}(\alpha)$. However, then we have $K' \prec_\alpha K$ – contradicting the minimality of K. □

Claim 3. *Let $\tau_1, \tau_2 \in \mathrm{top}(\alpha)$ be two topological orderings starting with $K_1, K_2 \in \Pi(G)$, respectively. If $K_1 \prec_\alpha K_2$ then $K_1 \cap K_2 = S \in \mathrm{FP}(\iota^{-1}(K_2), \mathcal{T})$.*

Proof. Since K_1 and K_2 correspond to $\tau_1, \tau_2 \in \mathrm{top}(\alpha)$, we have $K_1 \cap K_2 = S \in \Pi(G) \cup \Delta(G)$ by Lemma 5 – in fact, S is a prefix of τ_1 and τ_2. As we assume $K_1 \prec_\alpha K_2$, we have $K_1 \neq K_2$ and, thus, $S \in \Delta(G)$. Let $F_1, F_2 \in \mathcal{B}(S)$ be the S-flowers containing K_1 and K_2, respectively. The order $K_1 \prec_\alpha K_2$ implies $F_1 \prec_T F_2$ (item (iii) in the definition of $\prec_\alpha$), meaning that F_1 contains some node of (T, r, ι) that is on the unique path from F_2 to the root of T. But then, by the definition of S-flowers and Lemma 9, the first edge on this path that leads to a node in F_1 connects two nodes x, y with $\iota(x) \cap \iota(y) = S$. Hence, $S \in \mathrm{FP}(\iota^{-1}(K_2), \mathcal{T})$. □

This completes the proof of Proposition 3. □

Claim of Theorem 2. *For an input UCCG G, Algorithm 2 returns the number of AMOs of G.*

Proof. Observe that recursive calls are performed in line 13 if $\mathcal{C}_G(\iota(v)) \neq \emptyset$. The only graphs with $\mathcal{C}_G(S) = \emptyset$ for all $S \in \Pi(G)$ are the complete graphs, i. e., the graphs with $|\Pi(G)| = 1$. We have $|\Pi(H)| < |\Pi(G)|$ for all graphs G and $H = G[V \setminus S]$ with $S \in \Pi(G)$. Hence, we may assume by induction over $|\Pi(G)|$ that the subproblems are handled correctly – the base case being given by complete graphs.

The correctness of the algorithm follows from Proposition 3, as it traverses the clique tree with a BFS in order to compute the sets $\mathrm{FP}(v, \mathcal{T})$ and evaluate this formula. □

Proofs of Section 6: The Complexity of #AMO

Recall that for $H \in \mathcal{C}_G(\pi(K))$, we defined $P_{\text{out}}(H)$ to be the set of preceding neighbors of the vertices of H at state $\mathcal{X}_{\text{out}}$ when H was output during Algorithm 3. Now, we consider $H \in \mathcal{C}_G(K)$. We use the same notation to refer to the preceding neighbors in the analogously defined state in Algorithm 1. In fact, as $P_{\text{out}}(H)$ is independent of the permutation $\pi(K)$ (Proposition 1), it will be the same set.

We denote the previously visited vertices at $\mathcal{X}_{\text{out}}$, which are not in $P_{\text{out}}(H)$, by W, i. e., $W = V(\mathcal{X}_{\text{out}}) \setminus P_{\text{out}}(H)$.

⋆ **Lemma 15.** *Let G be a chordal graph, $K \in \Pi(G)$, and $H \in \mathcal{C}_G(K)$. Then, $P_{\text{out}}(H)$ separates V_H from $W = V(\mathcal{X}_{\text{out}}) \setminus P_{\text{out}}(H)$ and is a minimal separator of G.*

Proof. The set $P_{\text{out}}(H)$ is a proper subset of all previously visited vertices (V_H is not part of the *maximal* clique K Algorithm 1 starts with). Since $P_{\text{out}}(H)$ contains all visited neighbors of V_H, it separates V_H from W. To see this, assume for sake of contradiction that there is a path from $v \in V_H$ to $w \in W$ in G without a vertex in $P_{\text{out}}(H)$. Consider the shortest such path and let y be the first vertex with successor z preceding it in the LBFS ordering produced by Algorithm 1: $v - \cdots - x - y - z - \cdots - w$. Then $\{x, z\} \in E_G$, as the LBFS computes a PEO. Hence, the path is not the shortest path and, thus, y cannot exist. Since there can be no direct edge from v to w, the set P_{out} is indeed a separator.

We prove that there is a vertex in W, which is a neighbor of all vertices in $P_{\text{out}}(H)$. Consider the vertex in $P_{\text{out}}(H)$, which is visited last (denoted by p). When vertex p is processed, it has to have a neighbor $x \in W$, which was previously visited, else p would be part of H. This is because the

preceding neighbors would be identical to the ones of the vertices in H (i. e., $\mathcal{P}_{\text{out}}(H) \setminus \{p\}$), meaning by Lemma 12 that p would be in the same set in $\mathcal{S}$. It would follow that either p and the vertices in H are appended to L when p is visited or were already appended to L previously. In both cases, p would be in V_H, which is a contradiction.

Hence, such a vertex x has to exist. Moreover, x has to be connected to all vertices in $P_{\text{out}}(H)$ because of the PEO property (all preceding neighbors of a vertex form a clique).

From the first part of the proof, we know that x and $y \in H$ are separated by $P_{\text{out}}(H)$. As both x and y are fully connected to $P_{\text{out}}(H)$, it follows that this set is also a *minimal* $x - y$ separator.

□

⋆ **Lemma 16.** *Let G be a chordal graph for which the number of AMOs is computed with the function* `count` *in Algorithm 2. Let H be any chordal graph for which* `count` *is called in the recursion (for $H \neq G$). Then $V_H = F \setminus S$ for some S-flower F in G with $S \in \Delta(G)$.*

Proof. Let S_H be the union of all sets $P_{\text{out}}(\tilde{G})$ for $\tilde{G}$ on the recursive call stack from the input graph G to currently considered subgraph H. We define $P_{\text{out}}(G) = \emptyset$ for convenience.

Let $H \neq G$, we show by induction that (i) S_H is a minimal separator in G, (ii) S_H is fully connected to V_H in G, and (iii) $V_H = F \setminus S_H$ for some S_H-flower F.

In the base case, $H \in \mathcal{C}_G(K)$ for some clique $K \in \Pi(G)$. By Lemma 15, S_H is a minimal separator in G, which is by definition connected to all vertices in H. Hence, as H is connected, $V_H \subseteq F \setminus S_H$ holds for an S_H-flower F. We show the equality by contradiction. Assume there is a vertex $v \in F \setminus S$ but not in V_H. Then v can neither be a vertex in W nor the neighbor of a vertex in W, as by the definition of flowers there has to be a path from v to V_H in $G[V_G \setminus S_H]$ – this would violate that V_H is separated from W by S_H (Lemma 15). Moreover, v is a neighbor of all vertices in S_H. Hence, we have $P_{\text{out}}(v) = P_{\text{out}}(H) = S_H$ and $v \in V_H$. A contradiction.

Assume `count` is called with a graph $H \in \mathcal{C}_{G'}(K)$ for some graph G' and $K \in \Pi(G')$. By induction hypothesis, we have that $S_{G'}$ is a minimal separator in G and fully connected to $V_{G'}$. Moreover, $V_{G'} = F' \setminus S_{G'}$ for some F'-flower of $S_{G'}$. Now, $P_{\text{out}}(H)$ is by Lemma 15 a minimal separator in G' for some vertices x and y. As x and y are connected to every vertex in $S_{G'}$, it follows that $S_H = S_{G'} \cup P_{\text{out}}(H)$ is a minimal x-y separator in G. Furthermore, S_H is fully connected to V_H in G and it can be easily seen that $V_H \subseteq F \setminus S_H$. To show equality, observe that every vertex v in $F \setminus S_H$ is in $V_{G'}$ (if it is not separated from V_H by S_H in G, it is clearly not separated from V_H by $S_{G'}$ in G, hence v is in $F' \setminus S_{G'} = V_{G'}$). Thus, the same argument as in the base case applies and the statement follows. □

Claim of Proposition 4. *Let G be a UCCG. The number of distinct UCCGs explored by* `count` *is bounded by $2|\Pi(G)| - 1$.*

Proof. By Lemma 16, it remains to bound the number of flowers in G. Each flower is associated with a minimal separator S and there are at most $|\Pi(G)| - 1$ such separators, as they are associated with the edges of the clique tree (Blair and Peyton 1993). Let r (which is initially $|\Pi(G)| - 1$) be an upper bound for the number of remaining separators. Now consider separator S. If $\mathcal{B}(S)$ has k flowers, S can be found on at least $k - 1$ edges of the clique tree, namely the edges between the flowers (by Proposition 9 the flowers partition the bouquet and by definition of flowers, the intersection of cliques from two S-flowers has to be a subset of S). Thus, we have at most $r - (k - 1)$ remaining separators. The maximum number of flowers is obtained when the quotient $k/(k-1)$ is maximal. This is the case for $k = 2$. It follows that there are at most $2(|\Pi(G)| - 1)$ flowers.

When bounding the number of explored UCCGs, we additionally take into account the input graph and obtain as bound $2(|\Pi(G)| - 1) + 1 = 2|\Pi(G)| - 1$. □

Claim of Theorem 3. *The Clique-Picking algorithm runs in time $\mathcal{O}\big(\,|\Pi(G)|^2 \cdot (|V| + |E|)\,\big)$.*

Proof. By Proposition 4, `count` explores $\mathcal{O}(|\Pi(G)|)$ distinct UCCGs. For each of them, the clique tree is computed in time $\mathcal{O}(|V| + |E|)$. Afterwards, for each maximal clique, the subproblems are computed by Algorithm 1 in time $\mathcal{O}(|V| + |E|)$ by Theorem 1.

For the computation of $\phi(S, \text{FP}(v, \mathcal{T}))$, note that FP can be computed straightforwardly: Traverse the clique tree with a BFS, keep track of the nodes on the path from root r to any visited node, compute FP with its definition.

The function ϕ can be evaluated with the formula from Lemma 6. There are $\mathcal{O}(|S|)$ subproblems and for each a sum over $\mathcal{O}(|S|)$ terms has to be computed (as l is always smaller than $|S|$). Because S is a clique, the effort is in $\mathcal{O}(|E|)$. □

With $\text{Pe}(K, \text{FP})$ we denote the set of permutations of clique K without a prefix in FP and with $\#\text{Pe}(K, \text{FP})$ its size (which coincides with $\phi(K, \text{FP})$).

⋆ **Lemma 17.** *Algorithm 4 samples a permutation $\pi \in \text{Pe}(K, \text{FP})$ uniformly at random.*

Proof. We show this by induction over the size of the clique K. In the base case, we have a singleton $K = \{v\}$ and $\pi = (v)$ will be chosen with probability $1 = 1/\#\text{Pe}(K, \text{FP})$.

Let K be a clique with $|K| > 1$. Then the algorithm will choose a vertex v and, recursively, find a uniform permutation for the remaining clique $\tilde{K} = K \setminus v$ (regarding new forbidden prefixes $\tilde{\text{FP}}$). This $\tilde{\text{FP}}$ contains only sets $X_i, \ldots, X_\ell \in \text{FP}$ that contain v, as the other forbidden prefixes cannot occur with v being picked first. Hence,

$$\tilde{\text{FP}} = \{X_i \setminus v, \ldots, X_\ell \setminus v\}$$

with i being the smallest index such that $v \in X_i$. For the probability $\Pr(v_\pi \mid K, \text{FP})$ that the first vertex v is chosen according to a permutation π (given some K and FP), it holds that:

$$\Pr(v_\pi \mid K, \text{FP}) = \frac{\#\text{Pe}(K \setminus v_\pi, \tilde{\text{FP}})}{\#\text{Pe}(K, \text{FP})}.$$

input : Clique K, list $\mathrm{FP} = (X_1, X_2, \ldots, X_\ell)$ with $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$.
output: Permutation π.

function `drawperm`(K, FP, memo)
  foreach $v \in K$ **do**
    if v *in* X_ℓ **then**
      Let i be smallest such that $v \in X_i$.
      $\mathrm{wt}(v) \leftarrow \phi(K \setminus v, \{X_i \setminus v, \ldots, X_\ell \setminus v\})$
    else
      $\mathrm{wt}(v) \leftarrow \phi(K \setminus v, \emptyset)$
    end
  end
  $v \leftarrow$ `drawvertex`(wt)
  if v *in* X_ℓ **then**
    Let i be smallest such that $v \in X_i$.
    $\tilde{\mathrm{FP}} \leftarrow \{X_i \setminus v, \ldots, X_\ell \setminus v\}$
  else
    $\tilde{\mathrm{FP}} \leftarrow \emptyset$
  end
  if $|K \setminus v| = 0$ **then**
    return (v)
  else
    return `concat`$(v,$ `drawperm`$(K \setminus v, \tilde{\mathrm{FP}}))$
  end
end

Algorithm 4: The algorithm returns a uniform permutation of clique K under the constraint that the permutation does not start with a set in FP.

This is due to the fact that `drawvertex` samples v proportional to the weights $\mathrm{wt}(v)$, which is the number of permutations without forbidden prefix in $\tilde{\mathrm{FP}}$ that start with v.

Now, we want to compute $\Pr(\pi \mid K, \mathrm{FP})$, i. e., the probability that the vertices in K with forbidden prefixes FP are permuted according to (some not forbidden) π. For this, we make use of the induction hypothesis:

$$\Pr(\tilde{\pi} \mid K \setminus v_\pi, \tilde{\mathrm{FP}}) = \frac{1}{\#\mathrm{Pe}(K \setminus v_\pi, \tilde{\mathrm{FP}})}$$

with $\tilde{\pi}$ being the remaining part of permutation π after removing v_π. Note in particular that $\tilde{\mathrm{FP}}$ is valid because

$$X_i \setminus v_\pi \subset X_{i+1} \setminus v_\pi \cdots \subset X_\ell \setminus v_\pi$$

and hence the induction hypothesis applies. We conclude:

$$\begin{aligned}\Pr(\pi \mid K, \mathrm{FP}) &= \Pr(v_\pi \mid K, \mathrm{FP}) \cdot \Pr(\tilde{\pi} \mid K \setminus v_\pi, \tilde{\mathrm{FP}}) \\ &= \frac{\#\mathrm{Pe}(K \setminus v_\pi, \tilde{\mathrm{FP}})}{\#\mathrm{Pe}(K, \mathrm{FP}) \cdot \#\mathrm{Pe}(K \setminus v_\pi, \tilde{\mathrm{FP}})} \\ &= \frac{1}{\#\mathrm{Pe}(K, \mathrm{FP})}.\end{aligned}$$

□

Claim of Theorem 4. *There is an algorithm that, given a connected chordal graph G, uniformly samples AMOs of G in time $\mathcal{O}(|V| + |E|)$ after an initial $\mathcal{O}(\Pi(G)^2 \cdot |V| \cdot |E|)$ setup.*

input : A UCCG G.
output: AMO of G.

function `sample`(G, memo)
  $(K, \mathrm{FP}, \mathcal{C}_G(K)) \leftarrow$ `drawclique`(memo$[G]$)
  $\tau \leftarrow$ `drawperm`$(K, \mathrm{FP}, \mathrm{memo})$
  foreach $H \in \mathcal{C}_G(K)$ **do**
    $\tau \leftarrow$ `concat`$(\tau,$ `sample`$(H, \mathrm{memo}))$
  end
  return τ
end

memo $\leftarrow$ `precount`(G)
$\tau \leftarrow$ `sample`(G, memo)
return AMO of G given by τ

Algorithm 5: The algorithm uniformly samples an AMO from a UCCG G. A modified version of Clique-Picking (`precount`) has to be executed on G in advance.

Proof. We first prove that Algorithm 5 samples uniformly and will take care of the run time afterwards.

Denote with $\Pr(\tau_\alpha(G))$ the probability that we draw a topological ordering τ of the vertices in G that represents α. We show the theorem by induction, similarly to the proof of Theorem 2. For the base case of a single clique K, the algorithm returns a uniformly sampled AMO, as FP is empty and `drawperm` returns a uniformly sampled permutation. It follows that for each AMO α:

$$\Pr(\tau_\alpha(G)) = 1/|K|! = 1/\#\mathrm{AMO}(G).$$

For an arbitrary graph G, let K_α be the unique clique at which α is considered (such a clique exists by the proof of Proposition 3) with forbidden prefixes FP, and π_α the corresponding permutation of K_α. The term $\Pr(K_\alpha \mid \mathrm{FP})$ describes the probability that K_α is drawn:

$$\begin{aligned}&\Pr(\tau_\alpha(G)) \\ &= \Pr(K_\alpha \mid \mathrm{FP}) \Pr(\pi_\alpha \mid K_\alpha, \mathrm{FP}) \prod_{H \in \mathcal{C}_G(K_\alpha)} \Pr(\tau_{\alpha[H]}(H)) \\ &= \frac{\#\mathrm{Pe}(K_\alpha, \mathrm{FP}) \prod_{H \in \mathcal{C}_G(K_\alpha)} \#\mathrm{AMO}(H)}{\#\mathrm{AMO}(G) \cdot \#\mathrm{Pe}(K_\alpha, \mathrm{FP}) \prod_{H \in \mathcal{C}_G(K_\alpha)} \#\mathrm{AMO}(H)} \\ &= \frac{1}{\#\mathrm{AMO}(G)}.\end{aligned}$$

For the second step, we make use of the following facts. By induction hypothesis we have

$$\prod_{H \in \mathcal{C}_G(K_\alpha)} \Pr(\tau_{\alpha[H]}(H)) = \frac{1}{\prod_{H \in \mathcal{C}_G(K_\alpha)} \#\mathrm{AMO}(H)};$$

and

$$\Pr(K_\alpha \mid \mathrm{FP}) = \frac{\#\mathrm{Pe}(K_\alpha, \mathrm{FP}) \cdot \prod_{H \in \mathcal{C}_G(K_\alpha)} \#\mathrm{AMO}(H)}{\#\mathrm{AMO}(G)}$$

as K_α is picked with probability proportional to the number of AMOs that are considered at clique K_α (function `drawclique` in Algorithm 5); and by Lemma 17:

$$\Pr(\pi_\alpha \mid K_\alpha, \mathrm{FP}) = \frac{1}{\#\mathrm{Pe}(K_\alpha, \mathrm{FP})}.$$

We discuss the run time of Algorithm 5. Assuming a sampling generator has been initialized during `precount`, the function `drawclique` can be performed in $\mathcal{O}(1)$ with the Alias Method (Vose 1991). Assuming, furthermore, pointers to the subproblems $H \in \mathcal{C}_G(K)$, there is no additional effort in obtaining those. Since only $\mathcal{O}(|V|)$ subproblems are considered, these steps take $\mathcal{O}(|V|)$ time in total.

Hence, `drawperm` is the most expensive step and it remains to bound its cost. For each place in the permutation π, which is build, the algorithm goes once through all vertices v and assigns the values wt. We assume that the values for ϕ are precomputed. This is feasible as ϕ only depends on two parameters: the size of the current K and the index i of the remaining forbidden prefixes $X_i, \ldots, X_\ell$. Assuming such precomputations, `drawperm` can be implemented in time $\mathcal{O}(|K|^2)$ for a clique K. In total, the run time is bounded by $\mathcal{O}(|V| + |E|)$ (each vertex is placed once in a permutation π and the cost are bounded by $|K|$, which is linear in the number of neighbors of this vertex).

For the precomputation of ϕ, it is possible to calculate all ϕ-values in time $\mathcal{O}(|V| \cdot |E|)$ by dynamic programming (similarly as in the proof of Theorem 3). To see this, note that ϕ can also be expressed as:

$$\phi(S, R) = |S|! - \sum_{i=1}^{\ell} |X_i|! \cdot \phi(S \backslash X_i, \{X_{i+1} \backslash X_i, \ldots, X_\ell \backslash X_i\}).$$

Hence, we have, for each subproblem and clique, an effort of $\mathcal{O}(|V| \cdot |E|)$. Since ϕ is the costliest precomputation, we conclude that preprocessing takes in total:

$$\mathcal{O}(|\Pi(G)|^2 \cdot |V| \cdot |E|). \qquad \square$$

We note that, while preprocessing requires an additional factor $|V|$ compared to Clique-Picking in theory, we observed that the sets FP are usually so small that its influence is negligible for most practical cases.

Claim of Theorem 5. *The problems* #AMO *and uniform sampling from a Markov equivalence class are in* P.

Proof. We begin by proving the statement for #AMO. It is well-known that this problem reduces to counting AMOs of UCCGs (He, Jia, and Yu 2015). This is the problem the Clique-Picking algorithm solves. By Theorem 3, it performs polynomially many arithmetic operations. Since #AMO(G) is bounded above by $n!$ (let n be $|V|$), all operations run in polynomial time because the involved numbers can be represented by polynomially many bits:

$$n! \leq n^n = (2^{\log n})^n = 2^{n \cdot \log n}.$$

We analyze the complexity of the uniform sampling problem. By Theorem 4, sampling is possible in linear time assuming a modified version of Clique-Picking was performed as preprocessing step. This preprocessing can be performed in polynomial time. Moreover, the bit complexity of sampling is still polynomial (by a similar argument as above). $\square$

Partially directed acyclic graphs (PDAGs) are, as the name suggests, partially directed graphs without directed cycles. *Maximally oriented PDAGs* (MPDAGs) are PDAGs such that *none* of the four Meek rules (Meek 1995) can be applied. Both graph types are usually used in graphical modeling to represent background knowledge.

Claim of Theorem 6. *The problem of counting the number of AMOs is* #P*-complete for PDAGs and MPDAGs.*

Proof. Note that, clearly, both problems are in #P.

We reduce the #P-hard problem of counting the number of topological orderings of a DAG (Brightwell and Winkler 1991) to counting the number of AMOs of a PDAG. Since we can exhaustively apply all Meek rules in polynomial time (Meek 1995) without changing the number of AMOs, this implies that the problem of counting AMOs is #P-complete for PDAGs and MPDAGs.

Given a DAG $G = (V, E)$, we construct the PDAG G' as follows: G' has the same set of vertices V as G and we add all edges from G to G'. We insert an undirected edge for all pairs of remaining nonadjacent vertices in G'.

Each AMO of G' can be represented by exactly one linear ordering of V (because G' is complete) and each topological ordering of G is a linear ordering as well. We prove in two directions that a linear ordering of V is an AMO of G' if, and only if, it is a topological ordering of G.

$\Rightarrow$) If a linear ordering τ represents an AMO of G', the edges in G are correctly reproduced. Hence, it is a topological ordering of G.

$\Leftarrow$) If a linear ordering τ is a topological ordering of G, the orientation of G' according to it is, by definition, acyclic and reproduces the directed edges in G'. As G' is complete, there can be no v-structures. Hence, τ represents an AMO of G'. $\square$

Notably, the reason Clique-Picking cannot be used to solve these counting problems can be directly connected to the hardness proof. Intuitively, these problems can be reduced to the setting that, when counting AMOs in UCCGs, some edge orientations in the chordal component are predetermined by *background knowledge*. Hence, in the Clique-Picking algorithm, when counting the number of permutations for a clique K, we have to count only those consistent with the background knowledge. But this is equivalent to the problem of counting the number of topological orderings.

Part II

Experimental Evaluation of the Clique-Picking Algorithm

We evaluated the practical performance of the Clique-Picking algorithm in a series of experiments. For this, we compared Clique-Picking, denoted (cp) in the following, with the state-of-the-art root picking algorithm *AnonMAO* (am) (Ganian, Hamm, and Talvitie 2020), a solver known as *TreeMAO* that performs dynamic programming on a tree decomposition (tw) (Talvitie and Koivisto 2019), and a recently proposed algorithm, which combines concepts from intervention design with dynamic programming, called *LazyIter* (li) (AhmadiTeshnizi, Salehkaleybar, and Kiyavash 2020). With these three competitors, we cover all currently known ways to approach the problem of counting Markov equivalent DAGs.

We omitted the classic root-picking algorithm (He, Jia, and Yu 2015) as well as MemoMAO (Talvitie and Koivisto 2019), as AnonMAO is an improved version of these algorithms, which has been shown to clearly outperform them in previous experiments (Ganian, Hamm, and Talvitie 2020).

All our experiments were performed on a desktop computer equipped with 14 GB of RAM and an Intel Core i7 X980 with 6 cores of 3.33 Ghz each. The system runs Ubuntu 20.04 LTS, 64bit. The Clique-Picking algorithm was compiled with `gcc 9.3.0` using `g++ -std=c++11 -O3 -march=native -lgmp -lgmpxx`. The dynamic program and the solver using tree decomposition were compiled with the same version of `gcc` and the makefiles provided by the authors. For the Python solver LazyIter we used `PyPy 7.3.1` (compiled with `gcc 9.3.0` for `Python 3.6.9`) in order to obtain a compiled and comparable version.

9 Generation of Random Chordal Graphs

As input for the solvers, we use various classes of random chordal graphs, which we explain in detail in the following. The results of our experiments can be found in Fig. 4 and Fig. 5. Beside the time used by the solvers, we also present the number of maximal cliques as well as the density $|E|/\binom{|V|}{2}$ that the graphs of the various random graph models have. In Fig. 3, the reader finds a cumulative plot over all experiments performed.

9.1 Random Subtree Intersections

The first family of random chordal graphs that we study is due to Seker et al. (2017). It is based on the characterization of chordal graphs as intersection graph of subtrees (take a tree, some subtrees of this tree, and build the intersection graph of these subtrees – the result is a chordal graph). The family is generated by an algorithm that obtains two parameters as input: n the number of vertices and k, a density parameter. In the first phase, a random tree on n vertices is generated with the algorithm of Rodionov and Choo (2004). Afterward, n subtrees are sampled randomly with the following procedure: Initialize the subtree to some random vertex v, then grow it by adding a random neighbor of the tree until the subtree reaches a size s randomly drawn from $\{1, \dots, 2k-1\}$. Finally, we take the intersection graph of these n subtrees, which is a chordal graph on n vertices.

We performed three experiments on this graph class. In each experiment we choose $n = 2^i$ for $i \in \{4, \dots, 12\}$ and k as either $k = \log n$ (Fig. 4.A), $k = 2 \log n$ (Fig. 4.B), or $k = \sqrt{n}$ (Fig. 4.C). In each case, we generated ten random graphs for every value of n and took the mean time used by the algorithms. As mentioned above, we also display the number of maximal cliques in the graph and the mean density of the graphs, i. e. $|E|/\binom{|V|}{2}$.

Our solver (cp) clearly performs best, solving even the largest instances with 4096 vertices in a short amount of time. This also holds for denser graphs, e. g. with $k = \sqrt{n}$. We can observe that the run time of (cp) only increases slowly for higher densities. The solver (am) can be clearly distinguished as second best, however, for larger and even moderately dense graphs the method timeouts, i. e., graphs with more than 256 vertices are infeasible for $k = 2 \log n$ and $k = \sqrt{n}$. The remaining methods (tw) and (li) perform comparably poorly and are only able to solve graphs with less than 100 vertices. Interestingly, the run time of both algorithms has an extremely high variance. For the case of (tw), this might be explained by the fact that the run time depends superexponentially on the size of the largest clique (denoted c) including a factor $c! \cdot 2^c \cdot c^2$.

9.2 Random Interval Graphs

The next class of random chordal graphs that we consider are *random interval graphs.* An interval graph is the intersection graph of a set of intervals. It is well-known that interval graphs are a subclass of chordal graphs and, thus, they are well-suited as input for our experiments. The solver (am) was proven to run in polynomial time on interval graphs (Ganian, Hamm, and Talvitie 2020), which makes for an interesting comparison.

We generate random interval graphs with the algorithm by Scheinerman (1988), which simply draws $2n$ random variables in $[0, 1]$, pairs these as intervals, and build the intersection graph of these intervals.

As in the previous section, we generated for every value $i \in \{4, \dots, 12\}$ ten random interval graphs on 2^i vertices. We ran all algorithms on these instances and built the mean of their runtime for each i. The results can be found in Fig. 4.D.

The instances of this test set are the densest ones that we considered during our experiments, with about two thirds of the possible edges being present in the graph. We observe that (tw) and (li) are only able to solve the smallest instances. The (am) algorithm manages to solve instances up to 128 vertices, but timeouts for larger ones. While, as mentioned above, the run time of (am) is bounded by a polynomial for this class of graphs, the degree of the polynomial appears to be too large for practical use. In contrast, the (cp) algorithm manages to solve *all* instances, though it has to be noted that it takes significant time (around 10 minutes) for the largest ones with 4096 vertices. An explanation for this is the higher

number of maximal cliques compared to the first set of experiments. Recall that the number of recursively explored subgraphs is bounded by $2 \cdot |\Pi(G)| - 1$. Since the graphs are dense, these subgraphs are also relatively large.

9.3 Random Perfect Elimination Orderings

Another characterization of chordal graphs is based on *perfect elimination orderings,* which we have discussed in Section 3. Recall that a perfect elimination ordering of a graph $G = (V, E)$ is a permutation π of its vertices such that for each $v \in V$ the set $\{ w \mid \pi(v) < \pi(w) \}$ is a clique. It is well-known that a graph is chordal if, and only if, it has a perfect elimination ordering (Blair and Peyton 1993). This leads to a simple randomized algorithm to generate chordal graphs: Start with a graph of n isolated vertices $v_1, \ldots, v_n$; iterate over the vertices in this order and for every v_i initialize a set $S = \{ v_j \mid v_j \in N(v_i) \wedge j > i \}$. Randomly choose a size s and, while $|S| < s$, add some further vertices v_j with $j > i$ to S; finally make S a clique and adjacent to v_i. This algorithm (as well as other refined versions of it) tends to produce quite dense graphs (Seker et al. 2017). Therefore, we use *constant size* bounds for the set S rather than a function that grows with n. More precisely, we run the algorithm with a parameter k and let it pick for every vertex a size bound $s \in \{k/2, \ldots, 2k\}$ at random. Note that this does *not* imply that the degree is bounded by $2k$.

We performed the same experiments as in the previous sections on this graph class for $k = 2$ and $k = 4$. The results can be found in Fig. 4.E and Fig. 4.F, respectively.

The instances generated with this method are quite interesting, because they have significantly different properties than, for example, the ones generated with the subtree intersection method. In particular, there is a large number of maximal cliques. Hence, these test sets contain some of the hardest instances for (cp). Still, the algorithm outperforms the other methods by a large margin. Only for the largest graphs with 4096 vertices and density parameter $k = 4$, there are some instances that (cp) could not solve within 30 minutes.

Notably, the (am) algorithm, which again takes the second place, performs worse on these instances compared to the ones generated with the subtree intersection method, too. For instance in Fig. 4.E the graphs are sparser than in Fig. 4.A, but (am) still takes more time; the same holds for Fig. 4.F and Fig. 4.B. This makes sense, as the number of subproblems of (am) was shown to depend on the size of the clique tree as well (Ganian, Hamm, and Talvitie 2020). The other two methods are less affected by the increase of maximal cliques and perform similar to the other experiments.

9.4 Random Tree Thickening

In this section, we mimic the experiments performed by He, Jia, and Yu (2015), Talvitie and Koivisto (2019), and Ganian, Hamm, and Talvitie (2020). He, Jia, and Yu (2015) proposed the following way of generating random chordal graphs with n vertices and kn edges, which we call *random tree thickening*. We start with a random tree on n vertices, which is generated by the Rodionov-Choo-Algorithm (Rodionov and Choo 2004). Then, as long as the graph in construction has less than kn edges, we repeatedly pick two non-adjacent vertices at random and connect them by an edge if the resulting graph is still chordal.

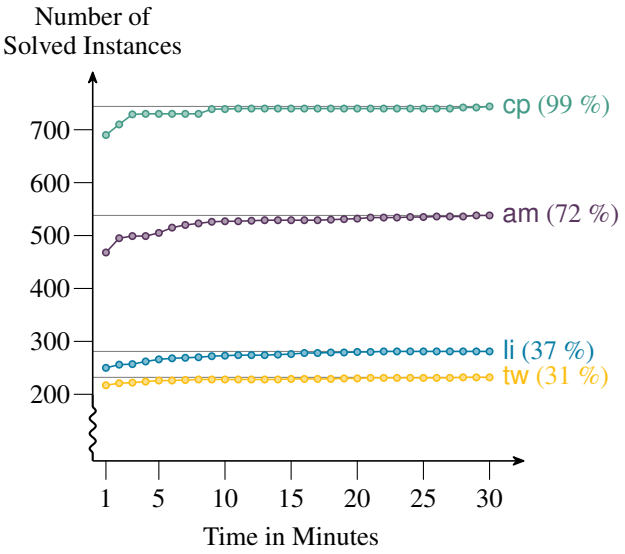


Figure 3: A cumulative distribution function plot that shows the performance of the four solvers Clique-Picking (cp), AnonMAO (am), TreeMAO (tw), and LazyIter (li) on all 750 random chordal graphs that where used for experiments in this section. The x-axis shows the time in minutes, and the y-axis the number of instances the solver solved (having x minutes per instance).

The advantage of this way of generating random chordal graphs is that we have fine control over the density of the generated graphs. However, the sketched algorithm is by far the slowest random chordal graph generator and becomes infeasible for larger values of n. We have therefore restricted the experiments in this sections to graphs with $n = 2^i$ vertices for $i \in \{4, \ldots, 10\}$. As density parameter k we used $k = 3$ (Fig. 5.A), $k = \log n$ (Fig. 5.B), and $k = \sqrt{n}$ (Fig. 5.C).

As the instances generated with this method are comparatively small and sparse, they are easy instances for (cp), which solves them all in seconds. For the other methods, the same insights as before hold.

10 Summary of the Experiments

The results of our experiments reveal that the Clique-Picking approach is superior to all state-of-the-art strategies for counting MAOs of chordal graphs by orders of magnitudes. In particular, the Clique-Picking algorithm even outperforms other solvers on instances on which they should naturally perform quite strong: It is not only faster on dense, but also sparse instances, and while (am) runs in polynomial time on interval graphs, it is still much slower than Clique-Picking on these graphs.

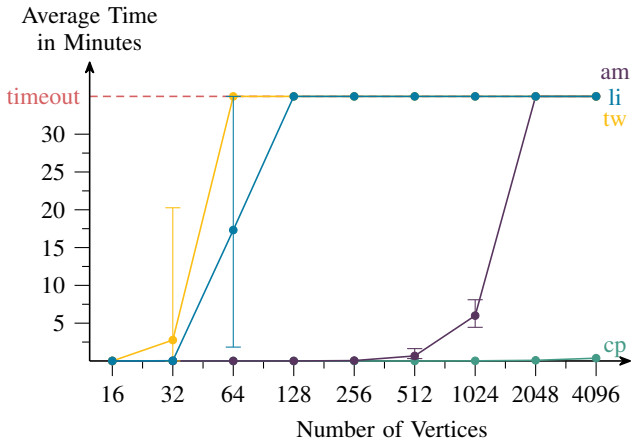
We summarize our experiments in Fig. 3, which contains a *cumulative distribution function plot* over all experiments that we performed. Overall. we produced 750 random

chordal graphs for the experiments in this section.

As seen in the previous sections, the solvers (tw) and (li) take the last places in our competition. It has to be noted that (li) even produced incorrect answers for a few instances (we double-checked these by brute-force enumeration). The (am) solver takes second place solving 72% of the instances, struggling in particular with larger and denser graphs.

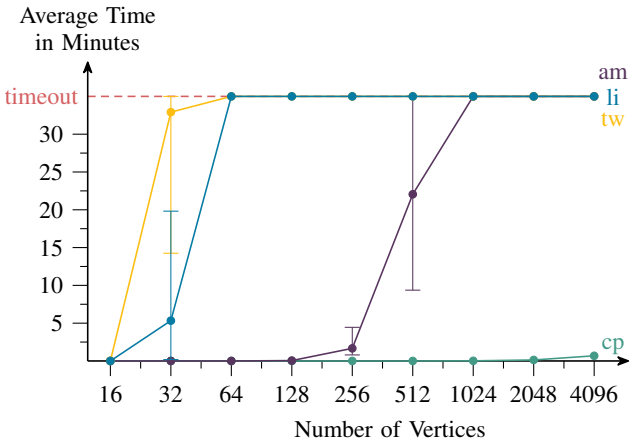
From the 750 graphs, there were only six which the Clique-Picking algorithm could not solve in 30 minutes (these were dense graphs generated by the random elimination ordering algorithm with $k = 4$, which have a relatively large number of maximal cliques). Moreover, (cp) was also extremely fast in solving most of the instances, almost 700 of them were solved in one minute or less (most of them in a few seconds), while the second-best method (am) solved significantly less instances even in the whole 30 minutes. This shows that the Clique-Picking algorithm is not only of theoretical value, but currently the by far most practical algorithm for counting Markov equivalent DAGs.

A Subtree Intersection, $k = \log n$



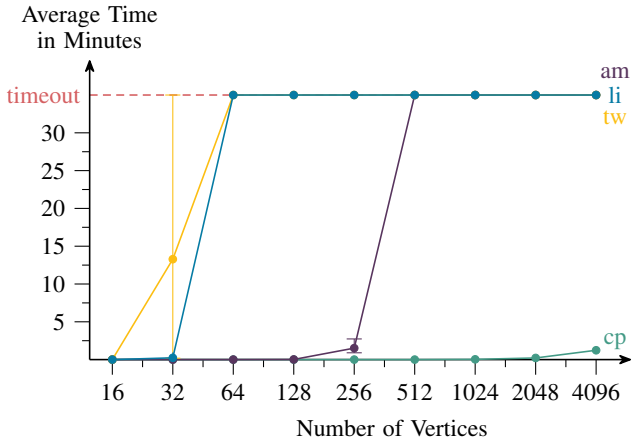
#cliques	6	10	19	33	66	126	231	441	841
density	0.50	0.40	0.31	0.22	0.14	0.09	0.05	0.03	0.02

B Subtree Intersection, $k = 2 \log n$



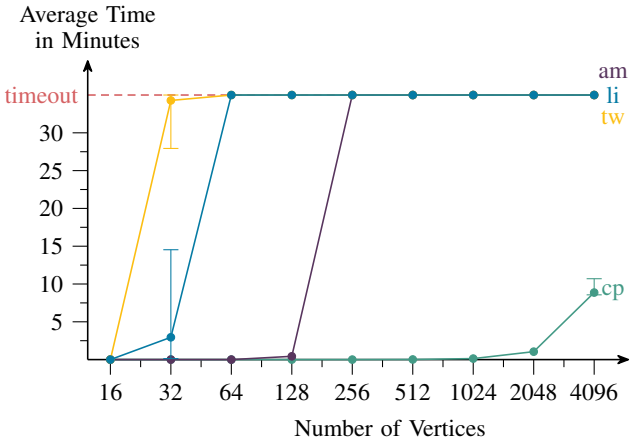
#cliques	4	7	13	25	45	80	155	296	576
density	0.86	0.75	0.62	0.46	0.34	0.23	0.16	0.10	0.06

C Subtree Intersection, $k = \sqrt{n}$



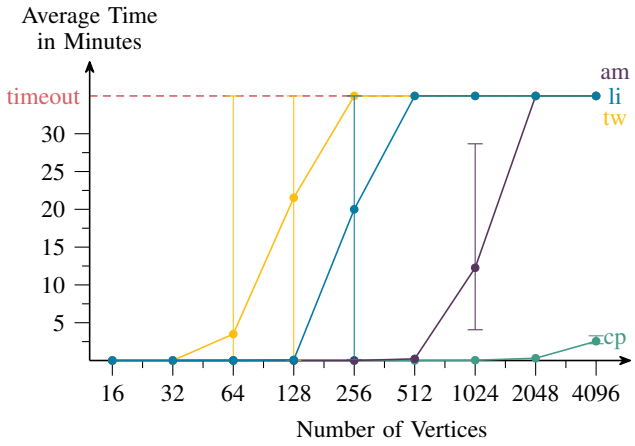
#cliques	5	9	17	28	46	73	124	202	334
density	0.63	0.53	0.41	0.41	0.33	0.34	0.28	0.25	0.22

D Random Interval Graphs



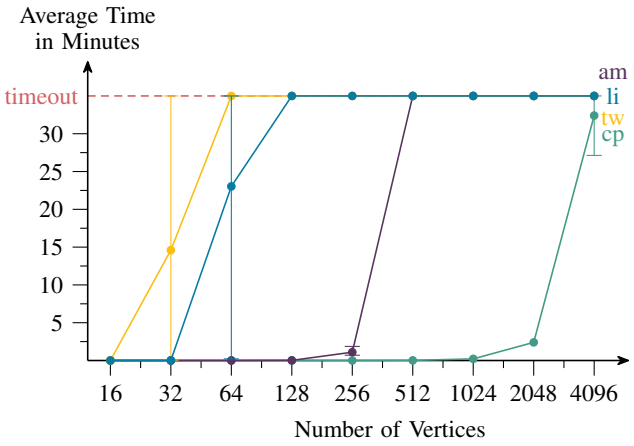
#cliques	6	11	21	46	86	168	340	688	1364
density	0.66	0.67	0.67	0.65	0.67	0.68	0.67	0.66	0.67

E Perfect Elimination Order, $k = 2$



#cliques	11	23	48	97	193	392	774	1554	3113
density	0.28	0.15	0.08	0.05	0.03	0.02	0.02	0.01	0.01

F Perfect Elimination Order, $k = 4$



#cliques	7	17	33	69	140	276	560	1121	2219
density	0.57	0.37	0.29	0.23	0.18	0.17	0.17	0.16	0.16

Figure 4: Comparison of the solvers Clique-Picking (cp), AnonMAO (am), TreeMAO (tw), and LazyIter (li). The plots show the mean time (dot) as well as the minimum and maximum time (thin lines) each solver needed over *ten* random graphs. A solver that requires more than 30 minutes or runs out of memory obtains a timeout, which contributes 35 minutes to the mean time. At the bottom of each plot we present the average number of maximal cliques as well as the average density.

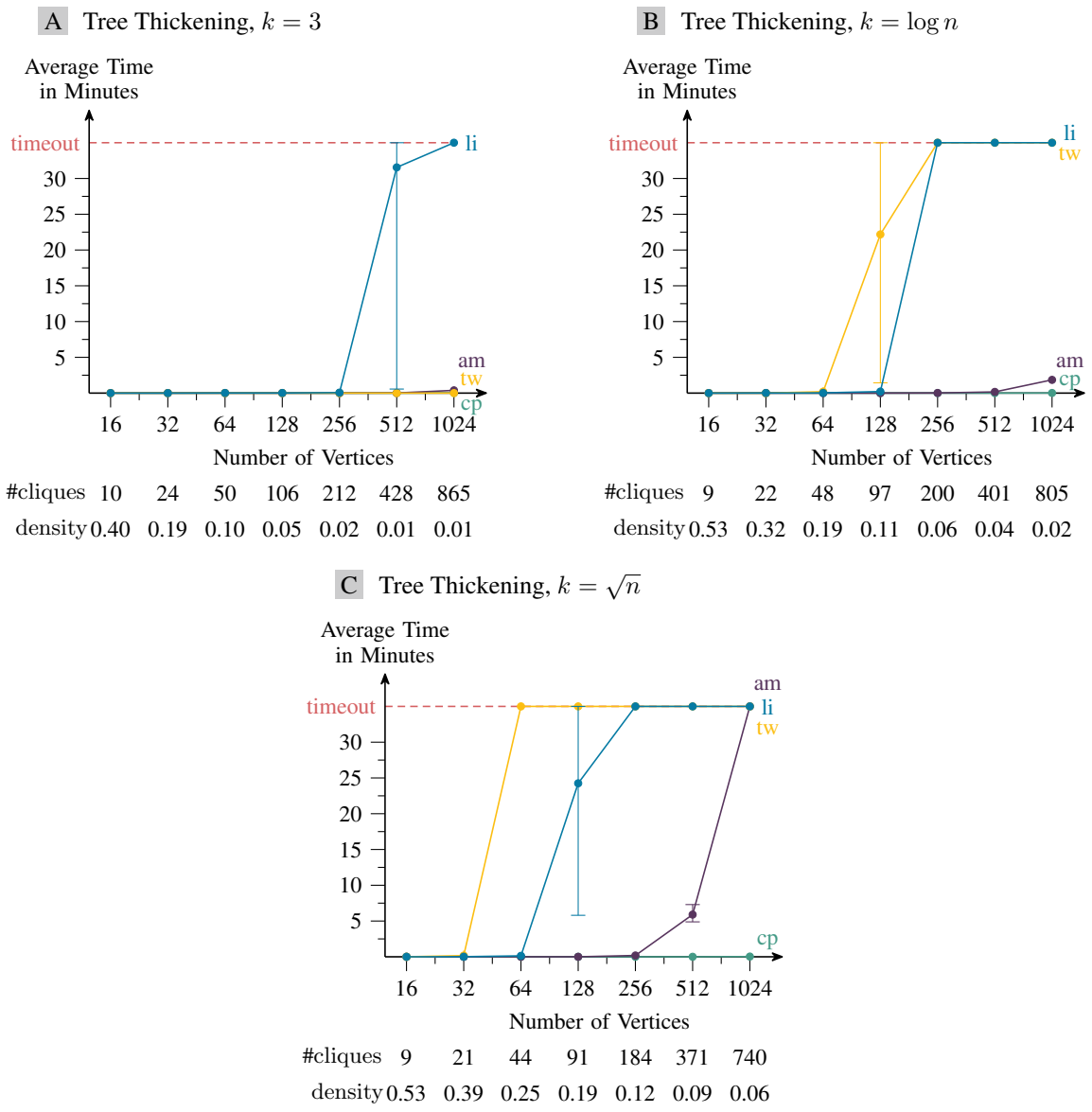


Figure 5: The same plots as in Fig. 4 for the random graphs from Section 9.4.