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 a 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 a MEC represented by a CPDAG may be superexponential in the number of vertices of the graph. More recently, He, Jia, and Yu (2015) proposed a strategy, in which the main idea was 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).

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. A 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 directed 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 a 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,

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

Therefore, the problem #AMO of counting the number of DAGs in a 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. 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:

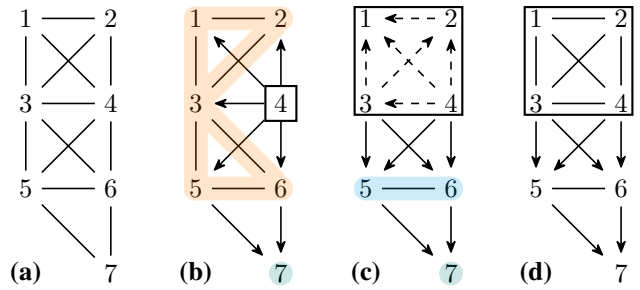


Figure 1: For a UCCG G in (a), the figure shows $G^{(4)}$ in (b), $G^{(4,3,2,1)}$ in (c), and $G^{\{1,2,3,4\}}$ in (d). The undirected components in $G^{(4)}$ and $G^{(4,3,2,1)}$ are indicated by the colored regions and the vertices put at the beginning of the topological ordering by a rectangle (all 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, 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 5. 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, which shows that $u \to v$ follows from iterative application of the first Meek rule. □

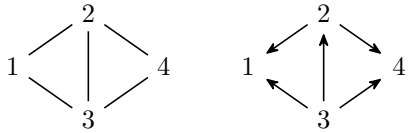
Both cases dealt with in the proof sketch can be seen in Fig. 1. The edge $5 - 6$ remains undirected as either vertex could be chosen first, while we have $5 \to 7$, because of the application of the first Meek rule to $\to 5 - 7$.

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. ◇

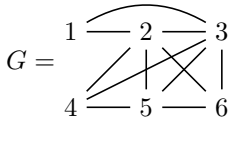
In this paper, we always consider sets $S \in \Delta(G) \cup \Pi(G)$ and collections $R \subseteq \Delta(G)$. Therefore, we can use the abbreviation $\phi(S) = \phi\big(S, \{\, S' \mid S' \in \Delta(G) \wedge S' \subsetneq S \,\}\big)$.

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:



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

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

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:*

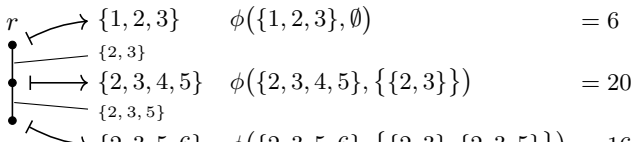
$$\#\mathrm{AMO}(G) = \sum_{v \in V_T} \phi(\iota(v), \mathrm{FP}(v,\mathcal{T})) \times \prod_{H \in \mathcal{C}_G(\iota(v))} \#\mathrm{AMO}(H).$$

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, function ι maps the nodes to the maximal cliques and the edges are associated with the 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 + prod · φ(ι(v), FP(v, T))
    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 that it runs in polynomial 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 $|\Pi(G)| - 1 - (k-1)$ other 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. $\square$

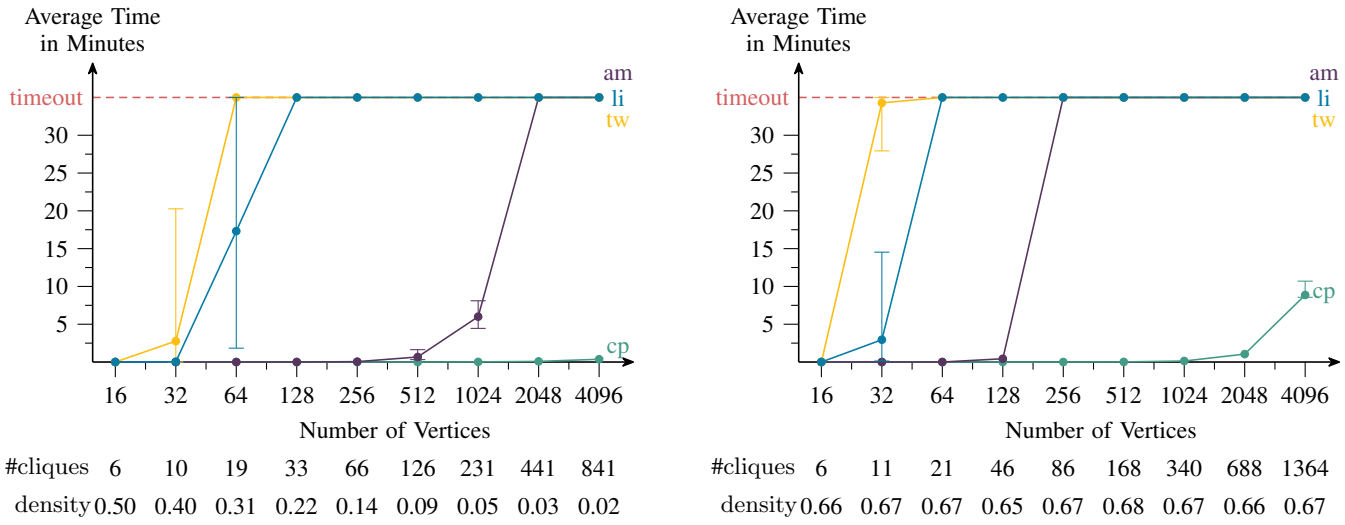


Figure 2: Experimental results for the solvers Clique-Picking (cp), AnonMAO (am), TreeMAO (tw), and LazyIter (li) on random chordal graphs with $n = 16, 32, \ldots, 4096$ vertices. For the left plot, we used graphs generated with the subtree intersection method and density parameter $k = \log n$; the 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 uniform 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 algorithm 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 the 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.

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.

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.

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.