Journal of Machine Learning Research 24 (2023) 1-45 Submitted 5/22; Revised 4/23; Published 7/23

Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs with Applications

Marcel Wienöbst M.WIENOEBST@UNI-LUEBECK.DE
Institute for Theoretical Computer Science
University of Lübeck
23538 Lübeck, Germany

Max Bannach MAX.BANNACH@ESA.INT
Advanced Concepts Team
European Space Agency
2201 AZ Noordwijk, The Netherlands

Maciej Liśkiewicz MACIEJ.LISKIEWICZ@UNI-LUEBECK.DE
Institute for Theoretical Computer Science
University of Lübeck
23538 Lübeck, Germany

Editor: Vanessa Didelez

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. As we show in experiments, these breakthroughs make thought-to-be-infeasible strategies in active learning of causal structures and causal effect identification with regard to a Markov equivalence class practically applicable.

Keywords: Causal inference, Graphical models, Markov equivalence, Interventions, Chordal graphs.

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 et al., 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 often unknown and cannot be identified unambiguously from observational data. Instead, the statistical properties of the data are shared 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 et al., 1995).

It is of great importance to investigate model learning and to analyze causal phenomena using MECs directly rather than the DAGs themselves. Consequently, Markov equivalence classes of DAGs constitute a central part of causal discovery and inference. Our work

©2023 Marcel Wienöbst, Max Bannach and Maciej Liśkiewicz.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v24/22-0495.html.

Table 1: An overview of the algorithmic improvements in the computation of the size of Markov equivalence classes. Here, n denotes the number of vertices, k indicates the size of the largest undirected clique in the graph, while d is the maximal number of undirected neighbors of a vertex. The first polynomial-time algorithm given in (Wienöbst et al., 2021b) is presented, applied and extended in this work. The unexplained abbreviations in the table should be read as follows: DP – Dynamic Programming; Dom. – Dominating.

	Approach	Complexity
Meek (1995)	Exhaustive search	$\mathcal{O}(n!)$
He et al. (2015); He and Yu (2016)	Root-Picking (RP)	$\mathcal{O}(n!)$
Talvitie and Koivisto (2019)	RP + Memoization (MM)	$\mathcal{O}(2^n \cdot n^4)$
Talvitie and Koivisto (2019)	DP on Clique-Tree	$\mathcal{O}(k!2^k k^2 n)$
Ghassami et al. (2019)	RP + MM + Clique-Tree	$\mathcal{O}(2^n \cdot n^4)$
Ganian et al. (2020, 2022)	RP + MM + Dom. Vertex	$\mathcal{O}(2^n \cdot n^4)$
AhmadiTeshnizi et al. (2020)	Intervention Design + MM	$\mathcal{O}(2^{n+d}(nd + d^3))$
Wienöbst et al. (2021b); This work	Clique-Picking + MM	$\mathcal{O}(n^4)$

contributes to this line of research by providing the first polynomial-time algorithms for *counting* and for *uniform sampling* Markov equivalent DAGs – important primitives in both theory and practice.

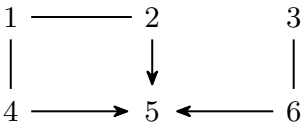
Finding the graphical criterion for two DAGs to be Markov equivalent (Verma and Pearl, 1990) and providing the graph-theoretic characterization of MECs as CPDAGs (Andersson et al., 1997) mark key turning points in this research direction. In particular, they have contributed to the progress of computational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

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 possibility of recovering true causal effects. Efficient algorithms for computing the size of an MEC are necessary, whenever researchers aim to quantify or even reduce the uncertainty present, which is the case particularly in causal effect identification over MECs (Maathuis et al., 2009) as well as active intervention design with the aim to recover the true DAG with as few experiments as possible (He and Geng, 2008; Hauser and Bühlmann, 2012; Shanmugam et al., 2015; Ghassami et al., 2018, 2019).

The first algorithmic approaches to counting the number of Markov equivalent DAGs date back to the work of Meek (1995). Therein, it is already observed that it suffices to consider the *undirected components*[1] of the CPDAG separately. Even when using exhaustive search for these components, this can already lead to reasonably fast algorithms when the CPDAG has few undirected edges, respectively small undirected components. In the worst-case of large undirected components, however, this approach requires exponential-time.

1. These are the connected components of the graph after removing all directed edges.

input : A CPDAG $C = (V, E)$.
output: Size of the MEC represented by C.
size $\leftarrow 1$
foreach *undirected component* G *of* C **do**
  size $\leftarrow$ size $\times$ Clique-Picking(G)
end
return size



Clique-Picking$(4 - 1 - 2)$
$\times$ Clique-Picking$(3 - 6)$
$3 \times 2 = 6$

Algorithm 1: The algorithm to compute the size of the MEC represented by a CPDAG C. Clique-Picking is presented as Algorithm 5 in Section 4. On the right, we show a simple example computation. The undirected component $4 - 1 - 2$ has three possible orientations as $4 \rightarrow 1 \leftarrow 2$ is disallowed (more on this in the subsequent section). In total, there are 6 DAGs in the MEC.

Starting with the work of He et al. (2015), the problem has been readdressed with better and better worst-case bounds of the run-time (see Table 1 for an overview). Particularly, the *root-picking* approach has been successively refined in multiple works (He et al., 2015; He and Yu, 2016; Talvitie and Koivisto, 2019; Ghassami et al., 2019; Ganian et al., 2020) bringing the time complexity down to $\mathcal{O}(2^n \cdot n^4)$, where n denotes the number of vertices, which still amounts to exponential worst-case time.

In this paper, we present the culmination of these algorithmic efforts, the first algorithm with polynomial time complexity $\mathcal{O}(n^4)$ for counting the number of Markov equivalent DAGs. Preliminary results of this work have been presented at the AAAI Conference on Artificial Intelligence, AAAI 2021 (Wienöbst et al., 2021b). Therein, the polynomial-time algorithm has been proposed and an implementation has been given, which outperforms the previous methods. This algorithm certifies that the problem can be solved efficiently – in theory and practice. It also implies a polynomial-time algorithm for uniformly sampling of DAGs from an MEC. A major focus of this work is on the further consequences of this breakthrough and, in particular, to illustrate its application to downstream tasks in causal discovery and inference, which have been underexplored thus far. To make such applications possible, we analyze the counting problem particularly for *interventional* MECs and show how the algorithmic achievements translate to this setting, which plays an important role in many fundamental tasks in causality, two of which we discuss in depth. Additionally, we provide the first practical implementation of a uniform sampling algorithm for the members of an MEC by extending and improving the results from (Wienöbst et al., 2021b).

Our Contributions

The main achievement of our paper is Algorithm 1, which is the first polynomial-time algorithm for counting Markov equivalent DAGs:

Theorem 1 (Main) *For an input CPDAG C, Algorithm 1 returns the size of the MEC represented by C in polynomial time in the size of the graph.*

The key component of this algorithm, which we coin *Clique-Picking*, computes the number of Markov equivalent DAGs separately for each undirected component of the CPDAG. In order to do so, it utilizes the fact that these components are *chordal* by exploring their clique-tree representation and by evaluating a non-trivial recursive counting function.[2]

With the recursion used in Clique-Picking, the problem of uniformly sampling a DAG from a Markov equivalence class can be solved in polynomial-time as well, by running an adapted version of Clique-Picking as a preprocessing step, after which sampling is possible in linear-time.[3]

Theorem 2 *There is an algorithm that, given a CPDAG C, uniformly samples a DAG in the MEC represented by C in expected linear time in the size of the graph after an initial polynomial-time preprocessing setup.*

Both algorithms are easy to implement and very fast in practice – outperforming previous approaches by a large margin in experimental evaluations (Wienöbst et al., 2021b). We complement our theoretical findings with optimized implementations in C++ and Julia to facilitate the application to real-world problems.

Particularly to this end, a special focus in this paper is devoted to two applications of the Clique-Picking algorithm: Active learning of causal DAGs (He and Geng, 2008) and the global-IDA algorithm for causal effect identification (Maathuis et al., 2009). We argue that it is desirable in both cases to compute the size of an MEC (or more precisely, an *interventional* MEC), a task which, as we show, can be efficiently solved using Clique-Plicking as a subroutine.[4] Previously, the task of counting the number of Markov equivalent DAGs was *avoided* by researchers in these fields due to its apparent intractability, leading to the prevalent use of heuristics at the cost of accuracy. We demonstrate that through the new methods developed in this paper, such heuristics are not needed anymore, as the size of an MEC can now be computed fast in practice and empirically validate this claim.[5]

Finally, we complete the complexity-theoretical study of the counting problem by investigating the more general problem of counting the number of DAGs with additional background knowledge. We show that this problem is intractable under common complexity-theoretical assumptions by connecting it to classical counting problems.

The paper is split into roughly two parts: In the first less-technical half, we formally introduce the problem of computing the size of an MEC and give the well-known reduction to a purely graph-theoretical problem (Section 2), whose solution (Clique-Picking) will be presented later in Section 4. Before this, we derive and analyze in Section 3 the two applications mentioned above. For these, Clique-Picking may be viewed as a black-box algorithm,

2. All technical terms are formally introduced in the subsequent section.
3. The method we propose in this work is an improvement over the original algorithm given in (Wienöbst et al., 2021b). The algorithm is simpler and the preprocessing step asymptotically more efficient by a factor n.
4. Both applications have been considered before (see Ghassami et al. (2019) and other works), but never from the point of view of interventional MECs. Instead the more general setting of counting with background knowledge has been used, which we show to be computationally intractable. In particular, utilizing interventional MECs in the global-IDA algorithm connects two disjoint subfields of causality.
5. The code for the experiments can be accessed at the address `https://github.com/mwien/counting-with-applications`.

which allows us to defer the technically demanding introduction and analysis of this algorithm to Section 4 in the second half of this paper. Building on these techniques, in Sec. 5, we give the polynomial-time algorithm for uniform sampling of Markov equivalent DAGs and afterwards generalize the results to the setting of additional background knowledge in Section 6. In the subsequent Section 7 we give conclusions. To improve readability, we move some technical proofs to Section 8.

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 this 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 $(v, u) \notin E_G$. In the latter case u is called a parent of v. 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 $N_G(u)$. A *clique* is a set of pairwise adjacent vertices. We denote the induced subgraph of G on a set $C \subseteq V$ by $G[C]$. The undirected components of a partially directed graph G are the connected components in the undirected graph one obtains after removing all directed edges from G.

In causality theory, DAGs are used as mathematical models to represent causal relations (Pearl, 2009). For a DAG $D = (V, E)$, the vertices $V = \{1, \dots, n\}$ represent the random variables $X = (X_1, \dots, X_n)$. A distribution f over X is *Markov* to D if it factorizes as

$$f(x_1, \dots, x_n) = \prod_{i \in V} f(x_i \mid pa_i(D)),$$

where $pa_i(D)$ denotes the values $x_{j_1}, \dots, x_{j_k}$ of the parents $Pa_i(D) = \{j_1, \dots, j_k\}$ of vertex i in D. Two DAGs D_1 and D_2 are *Markov equivalent* if for any positive distribution f, f is Markov to D_1 if, and only, if it is Markov to D_2.

Due to Verma and Pearl (1990), we know a graphical criterion to decide this relation: Two DAGs are Markov equivalent if, and only if, they have the same *skeleton* and the same *v-structures*. 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 $a \to b \leftarrow c$.

The Markov equivalence relation partitions the set of all DAGs into Markov equivalence classes (MECs), where we denote the MEC of a DAG D as $[D]$. An MEC can be represented by a CPDAG G (*completed partially directed acyclic graph*, also known as an *essential graph*), which is the *union graph* of the DAGs in the equivalence class it represents. When we speak of the union of a set of graphs $\{G_1 = (V, E_1), \dots, G_k = (V, E_k)\}$, we think of the graph $G = (V, \bigcup_{i=1}^{k} E_k)$. The MEC represented by G is denoted as $[G]$. The undirected components of a CPDAG are *undirected and connected chordal graphs* (UCCGs) (Andersson et al., 1997). In a chordal graph, every undirected cycle of lengths ≥ 4 contains a chord, that is an edge between two vertices of the cycle, which is not part of the cycle.

The problem this paper addresses is, generally speaking, the opposite direction w.r.t. the definition of a CPDAG above: Given a CPDAG G, we aim to compute $|\,[G]\,|$. To do so,

we introduce the following terms[6]: An *extension* of a partially directed graph G is obtained by replacing each undirected edge with a directed one.[7] It is called a *consistent* extension if it is acyclic and does not create a new v-structure not present in G (this ensures that for a CPDAG G, the set of consistent extensions constitutes $[G]$). We will denote the number of consistent extensions of graph G as $\#\mathrm{EXT}(G)$. Hence, if G is a CPDAG, then $\#\mathrm{EXT}(G)$ is the size of the corresponding Markov equivalence class. We also refer to the *computational problem* of counting the number of consistent extensions for a given partially directed graph as #EXT:

Problem 3 #EXT

Instance: *A partially directed graph $G = (V, E)$.*
Result: *The number of consistent extensions of G.*

By restricting the instances to graphs of a specific graph class, we derive the #EXT problem for this class. Naturally, of particular interest is the class of CPDAGs. In this paper, we also study the problem #EXT for interventional essential graphs, general PDAGs, as well as for MPDAGs. While these graph classes are formally defined and discussed later, we want to highlight an important difference: Counting for MPDAGs and PDAGs[8] is intractable (more precisely, we show that it is #P-hard). For interventional essential graphs, however, it is possible to perform this task in polynomial-time, as they share important properties with CPDAGs (this is formalized in the subsequent section).

To start with, it is a crucial fact that, for a CPDAG G, each UCCG of G can be oriented independently of the other UCCGs and the directed part of G (Andersson et al., 1997). This means that to obtain a consistent extension of G, it suffices to orient each UCCG without creating a cycle or a v-structure. In line with the literature, we term such orientations AMOs (acyclic moral orientations). In accordance with the notation above, we define $\#\mathrm{AMO}(H)$ as the number of AMOs of a connected chordal graph (i.e., UCCG) H. We illustrate the introduced terms in Fig. 2, where the UCCGs of the given CPDAG can be oriented independently to yield a consistent extension. Thus, we can conclude that for a CPDAG G the size of $|\,[G]\,|$ is equal to

$$\#\mathrm{EXT}(G) = \prod_{H \text{ is UCCG in } G} \#\mathrm{AMO}(H). \tag{1}$$

In other words, the problem #EXT 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). We tackle this purely graph-theoretical problem in Section 4 and derive the first polynomial-time algorithm for it. For the subsequent Section 3, which focuses on applications for our methods, it is sufficient to know that such an algorithm exists.

6. Most of them are introduced more generally for a "partially directed graph G", you may replace this with a "CPDAG G" during the first read.
7. We also use the term orientation in addition to extension in this paper, which more or less means the same thing. In accordance with the literature, we prefer to use orientation for undirected graphs and extension for partially directed graphs.
8. These classes are essentially equivalent when it comes to the time complexity of the counting task, as for every PDAG, there exists an MPDAG with the same consistent extensions and it can be computed in polynomial time.

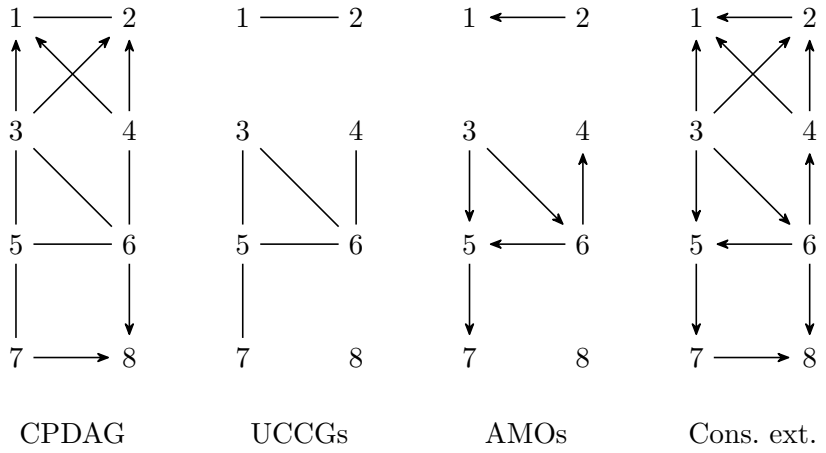


Figure 2: A CPDAG with its UCCGs and possible AMOs. Substituting these into the original CPDAG gives a consistent extension. The AMOs can be constructed independently of each other and the directed part of the CPDAG. Hence, the number of consistent extensions can be computed as the product of the number of AMOs for each UCCG.

3. Some Applications of Our Methods

The techniques developed in this paper can be applied to important tasks in causal discovery and inference. In this section, we highlight two possible applications: (i) to improve efficiency of learning from interventional data and (ii) estimating causal effects from an MEC representation.

3.1 Incorporating Observational and Interventional Data Efficiently

As discussed above, when dealing with purely observational data, a DAG is only identifiable up to its MEC (Andersson et al., 1997), which often makes it impossible to discover the unique structure. In some cases, however, additional experimental (also called interventional) data may be available or can be produced, in order to resolve the ambiguities. There is a large body of work in the field addressing this problem of estimating and explaining a causal structure from both observational *and* interventional data. Analogously to the observational case, all DAGs which satisfy the conditional independencies in both observational and interventional data form an equivalence class represented by an *interventional essential graph*. This graph (as the CPDAG for MECs) is formed by taking the union of the DAGs in this class. In the following, we define the concept more formally.

Let $D = (V, E)$ be a DAG and f be Markov to D. For a set of targets $I \subseteq V$, an intervention with perturbation targets $i \in I$ models the effect of replacing the observational distribution $f(x_i \mid pa_i(D))$ by $f^I(x_i)$ for all $i \in I$. The intervention graph of D is the DAG $D^I = (V, E^I)$, where $E^I = \{u \to v \in E \mid v \notin I\}$. Given a family of targets $\mathcal{I} \subseteq 2^V$ the pair $(f, \{f^I\}_{I \in \mathcal{I}})$ is $\mathcal{I}$-Markov to D if f is Markov to D and for all $I \in \mathcal{I}$ the interventional distribution f^I factors as

$$f^I(x_1, \ldots, x_n) = \prod_{i \notin I} f(x_i \mid pa_i(D)) \prod_{i \in I} f^I(x_i).$$

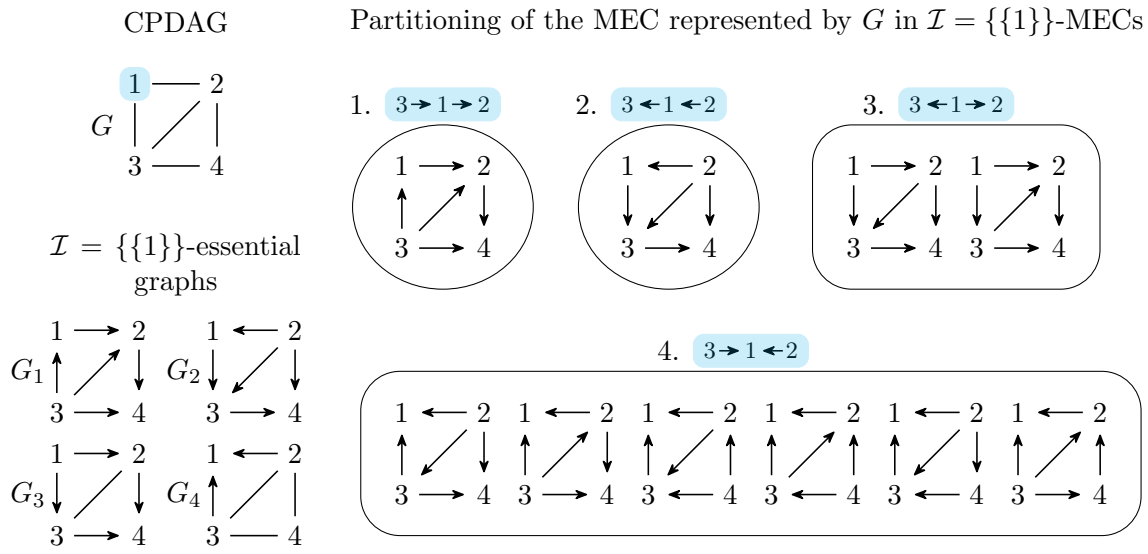


Figure 3: For the CPDAG G on the top left, we show on the right the interventional MECs for the family $\mathcal{I} = \{\{1\}\}$, i.e., an intervention is performed on the variable corresponding to vertex 1 (marked in color). The possible results for the intervention are shown in the colored regions and the DAGs are partitioned according to those configurations. Each $\{\{1\}\}$-MEC can be represented by the corresponding interventional essential graph on the bottom left, which encodes the still unknown edge orientations as undirected edges.

Two DAGs D_1 and D_2 are $\mathcal{I}$-Markov equivalent if for all positive distributions, $(f, \{f^I\}_{I\in\mathcal{I}})$ is $\mathcal{I}$-Markov for D_1 if and only if it is $\mathcal{I}$-Markov to D_2. This relation can be expressed in a graphical language as follows: For a conservative family of targets[9] $\mathcal{I}$, D_1 and D_2 are $\mathcal{I}$-Markov equivalent if for all $I \in \mathcal{I}$, D_1^I and D_2^I have the same skeleton and the same v-structures. The $\mathcal{I}$-Markov equivalence class of a DAG D ($\mathcal{I}$-MEC) is denoted by $[D]_\mathcal{I}$ and can be represented by the $\mathcal{I}$-essential graph $\mathcal{E}_\mathcal{I}(D) = \bigcup_{D'\in[D]_\mathcal{I}} D'$. A partially directed graph G is called an $\mathcal{I}$-essential graph if $G = \mathcal{E}_\mathcal{I}(D)$ for some DAG D. An example and further explanation regarding interventional MECs and essential graphs is given in Fig. 3.

The key property of interventional essential graphs, for our purposes, is that their undirected components are chordal and induced subgraphs, just as in CPDAGs:

Proposition 4 (Hauser and Bühlmann (2012)) *Let G be an $\mathcal{I}$-essential graph representing an $\mathcal{I}$-MEC $[D]_\mathcal{I}$ for a target family $\mathcal{I}$. Then, the undirected components of G are chordal (we will refer to them as UCCGs, just as for CPDAGs). Moreover, a DAG D' is in $[D]_\mathcal{I}$ if and only if D' can be obtained from G by acyclic moral orientations of the UCCGs of G independently of each other.*

For example, in the $\mathcal{I}$-essential graph G_4 in Fig. 3 representing the $\mathcal{I}$-MEC determined by the intervention result $3 \to 1 \leftarrow 2$, the undirected component is the triangle $2 \overset{\frown}{-} 3 - 4$.

9. A family of targets $\mathcal{I}$ is called conservative if for all $v \in V$, there is some $I \in \mathcal{I}$ such that $v \notin I$. Note that, e.g., any family containing the empty set $\emptyset$ is conservative. In this section we assume that the target families are conservative.

The $\mathcal{I}$-MEC consists of all six DAGs which can be obtained from G by acyclic moral orientations of this triangle. In general, this statement implies that for interventional essential graph G, the number of DAGs in the corresponding equivalence class $[D]_\mathcal{I}$ is

$$|[D]_\mathcal{I}| = \prod_{H \text{ is UCCG in } G} \#\mathrm{AMO}(H)$$

and, thus, this can be efficiently computed with the Clique-Picking algorithm. This leads to the following, main theorem of this section, which can be proved in the same manner as Theorem 1 and 2.

Theorem 5 *For a given interventional essential graph representing an $\mathcal{I}$-MEC $[D]_\mathcal{I}$, the number of DAGs in $[D]_\mathcal{I}$ can be computed in polynomial time. Moreover, sampling uniformly a DAG in $[D]_\mathcal{I}$ can be done in linear time, after preprocessing.*

The fact that the size of interventional essential graphs can be computed efficiently can be utilized in the context of *active learning* of the underlying causal DAG. It describes the process of designing experiments (i.e., interventions) in order to recover the DAG. A natural approach is to start estimating the essential graph (CPDAG) with observational data and afterwards, through experimentation, inferring the direction of beforehand unorientable edges to reduce the number of indistinguishable DAGs. Usually the objective is to find the underlying causal DAG with as few experiments as possible. Active learning has been the subject of a considerable amount of research, see Eberhardt et al. (2005); Eberhardt (2008); He and Geng (2008); Hauser and Bühlmann (2012); Hauser and Bühlmann (2014); Shanmugam et al. (2015); Ghassami et al. (2018); Greenewald et al. (2019); Squires et al. (2020) and the references therein.

One way of designing experiments is to use the following approach: Consider (for simplicity) only noiseless adaptive single-target interventions, i.e., each experiment manipulates a single variable of interest and the intervention informs us correctly about the resulting $\mathcal{I}$-MEC. In this setting, every intervention reveals the orientations of all edges adjacent to the intervened vertex and further edge orientations may be inferred by the Meek rules (Meek, 1995) (see Fig. 3 for an illustration). Additionally, we assume variables are manipulated sequentially, i.e., one can use intervention results obtained by manipulating the previous variables to select a current variable to intervene on. To choose the best intervention target, usually an objective function w.r.t. the current interventional essential graph is computed for each variable, often based on every possible intervention result.

Below we discuss three algorithms following this approach: MinMaxMEC and MaxEntropy by He and Geng (2008) and OptSingle by Hauser and Bühlmann (2014). The first two, particularly, use the sizes of the $\mathcal{I}$-MECs resulting from such hypothetical interventions, in order to compute the objective function. Hence, our methods are vital for the computational feasibility of those approaches. Moreover, we show that even the third approach can be sped up significantly.

The algorithms start with the (observational) MEC $[D]_\mathcal{I}$, i.e., with $\mathcal{I} = \{\emptyset\}$ for the true DAG D, which is represented as an $\mathcal{I}$-essential graph $\mathcal{E}_\mathcal{I}(D)$. Afterwards, while $|[D]_\mathcal{I}| > 1$, the current target family $\mathcal{I}$ and $G = \mathcal{E}_\mathcal{I}(D)$ are updated as follows: MinMaxMEC selects the variable to intervene on such that

$$v^* = \arg\min_{v \in V} \max_{D' \in [D]_\mathcal{I}} |[D']_{\mathcal{I} \cup \{\{v\}\}}|. \tag{2}$$

MaxEntropy chooses

$$v^* = \arg\max_{v \in V} H_v, \tag{3}$$

where H_v is the entropy defined as follows: Let $D_1, \dots, D_k \in [D]_{\mathcal{I}}$ be DAGs such that $[D_1]_{\mathcal{I}\cup\{\{v\}\}} \dot\cup \dots \dot\cup [D_k]_{\mathcal{I}\cup\{\{v\}\}}$ is a partition of $[D]_{\mathcal{I}}$. Then $H_v = -\sum_{j=1}^{k} \frac{l_j}{L} \log \frac{l_j}{L}$, with $l_j = |[D_j]_{\mathcal{I}\cup\{\{v\}\}}|$ and $L = |[D]_{\mathcal{I}}|$. Algorithm OptSingle computes a vertex

$$v^* = \arg\min_{v \in V} \max_{D' \in [D]_{\mathcal{I}}} \xi(\mathcal{E}_{\mathcal{I}\cup\{\{v\}\}}(D')), \tag{4}$$

where $\xi(H)$ denotes the number of undirected edges in a graph H. Next, the intervention on v^* is realized and the algorithm updates $G := \mathcal{E}_{\mathcal{I}\cup\{\{v^*\}\}}(D)$ and $\mathcal{I} := \mathcal{I} \cup \{\{v^*\}\}$ completing the iteration step. Note that none of these three strategies lead to an optimal algorithm (in a worst-case or average-case sense), but are effective greedy heuristics.

Example 1 *For the CPDAG G in Fig. 3, the algorithms MinMaxMEC, MaxEntropy and OptSingle partition, for every vertex v, the MEC represented by G into $\{\{v\}\}$-MECs according to all possible results for the intervention on v. The partitioning for $v = 1$ is shown in Fig. 3. The values needed to select $v^* = 2$ or $v^* = 3$ solving the Eq.* (2)*, Eq.* (3)*, resp. Eq.* (4)*, are given in the table below.*

v	cardinalities of $\{\{v\}\}$-essential MECs	max card.	H_v	number of undir. edges in $\{\{v\}\}$-essential graphs	max number
1	6,2,1,1	6	1.57	3,1,0,0	3
2	3,2,2,1,1,1	3	2.45	2,1,1,0,0,0	2
3	3,2,2,1,1,1	3	2.45	2,1,1,0,0,0	2
4	6,2,1,1	6	1.57	3,1,0,0	3

◇

Clearly, the most costly part of implementing MinMaxMEC and MaxEntropy is the counting of Markov equivalent DAGs. As this was previously thought infeasible, these methods were often avoided (Squires et al., 2020). However, one can easily see that, based on Theorem 5, the sizes of MECs needed to choose a vertex w.r.t. Eq. (2), resp. Eq. (3), can be computed in polynomial time, assuming the $(\mathcal{I} \cup \{\{v\}\})-$MECs are represented as interventional essential graphs.

Another efficiency issue of the algorithms, including OptSingle, concerns the computation of the interventional essential graphs for each possible intervention results (as there may be exponentially many such results and the algorithms consider every hypothetical result in advance, this step is crucial). Interestingly, using the ideas from Section 4 we can show that, given a current $\mathcal{I}$-interventional essential graph G and an interventional result on a vertex v, we can compute the new interventional essential graph in linear time. This is possible using an algorithm based on Maximum Label Search (Berry et al., 2009), which is also used in Clique-Picking, as we state in the theorem below.

The only thing left to be explained is how to enumerate the possible interventional results on v: To see this, let H be a UCCG of G containing v. Then the resulting orientations

of the incident undirected edges $u - v$ in G can be represented as a clique $K \subseteq N_H(v)$, which contains the incident vertices u of edges oriented as $u \to v$;[10] the edges $u - v$, with $u \in N_H(v) \setminus K$, are oriented as $u \leftarrow v$. Note, that K can be empty.

Theorem 6 *Assume D is a DAG, $\mathcal{I}$ is a target family, v is a vertex, and H is a UCCG of $G = \mathcal{E}_{\mathcal{I}}(D)$ containing v. Let a clique $K \subseteq N_H(v)$ represent orientations of edges $u - v$ in G as described above and let $D_K \in [D]_{\mathcal{I}}$ be a DAG with the edges oriented according to K. Then, given G, v, and K, the essential graph $G' = \mathcal{E}_{\mathcal{I}\cup\{\{v\}\}}(D_K)$ can be computed in time $\mathcal{O}(|V_G| + |E_G|)$.*

Notably, this theorem improves upon previous work by AhmadiTeshnizi et al. (2020), which gave an $O(d \cdot m)$ algorithm for this task (with d being the maximum degree of the graph). For example, computing a vertex solving Eq. (2) in MinMaxMEC can be implemented as shown in Algorithm 2 below. The other two approaches can be implemented similarly. The time complexity is $\mathcal{O}(\text{Val}(G) \cdot p(|V_G|, |E_G|))$, where Val(G) denotes the total number of intervention results and p is the polynomial bounding the time complexity of the Clique-Picking algorithm used to compute the size of an MEC.

input : An $\mathcal{I}$-essential graph $G = (V, E)$.
output: Vertex v^* solving Eq. (2).
if G is a DAG **then return** $\emptyset$ **and stop**
MinMax $\leftarrow \infty$
foreach *undirected component H of G* **do**
  foreach $v \in H$ **do**
    Max $\leftarrow 0$
    foreach *clique* $K \subseteq N_H(v)$ **do**
      for G, v, K compute $G' = \mathcal{E}_{\mathcal{I}\cup\{\{v\}\}}(D_K)$ using Theorem 6
      $c \leftarrow$ #EXT(G') using Algorithm 1
      if $c >$ Max **then** Max $\leftarrow c$
    end
    if Max $<$ MinMax **then**
      MinMax $\leftarrow$ Max; $v^* \leftarrow v$
  end
end
return v^*

Algorithm 2: An efficient implementation of MinMaxMEC: The algorithm computes a vertex solving Eq. (2).

These improvements can make the difference between infeasibility and practical applicability. We replicate the recent experimental results from Squires et al. (2020), which includes a comparison of the most popular algorithms for single-target adaptive active learning. For their experiments, a single chordal component was generated in two ways: Large and sparse graphs were sampled by adding edges to randomly generated trees, and small and very dense

10. The parents of v have to form a clique, else a new v-structure would be created, which would be in violation with the definition of $\mathcal{I}$-MECs.

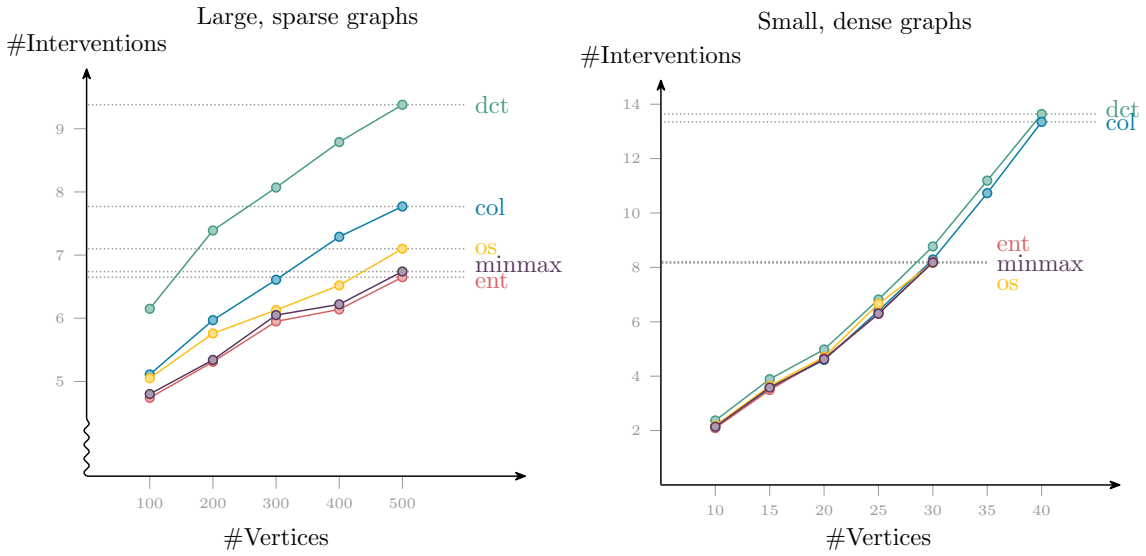


Figure 4: We compare the performance of the three discussed strategies MinMaxMEC (minmax), MaxEntropy (ent) and OptSingle (os), with the coloring-based strategy (col) of Shanmugam et al. (2015), and directed-clique-tree approach (dct) of Squires et al. (2020). On the left, the average number of interventions for large, sparse graphs; on the right, for small dense graphs. For each choice of parameters, 100 CPDAGs (chordal graphs) were generated as described in Squires et al. (2020). Afterwards, for each CPDAG, a DAG was uniformly chosen from the MEC (the true DAG, used as oracle for the intervention results).

graphs by "chordalizing" Erdös-Renyi graphs. The three strategies we discussed above were only included in the experiments on small graphs (between 8 and 14 vertices) due to their apparent infeasibility. We show that, using implementations[11] of our methods, these approaches scale to much larger graphs and that they deliver superior results compared to other algorithms.

In Fig. 4, the plot on the left shows the number of performed interventions of five active learning strategies for the large and sparse graphs and the right one for the small and very dense graphs. The three strategies we discussed above, MinMaxMEC, MaxEntropy and OptSingle, clearly outperform the other methods and, in particular, the two methods which utilize the sizes of the $\mathcal{I}$-MECs perform the best. In case of the sparse graphs the differences are larger due to the fact that the structure of the graph can be utilized to a higher degree; the dense graphs are close to fully connected graphs. Importantly, for the sparse graphs, where a lot of performance may be gained and which occur frequently in practice, the implementation using Clique-Picking is able to scale up to graphs with 500 vertices (taking about 10-15 minutes on a desktop computer for the largest graphs[12]). In case of the dense graphs, the implementations of MinMaxMEC, MaxEntropy and OptSingle can handle up

11. The experiments in this paper record the results using implementations in Julia. Additionally, in the code appendix (`https://github.com/mwien/counting-with-applications`), a C++implementation of Clique-Picking is provided.
12. The experiments were run on an Intel(R) Core(TM) i7-8565U CPU with 16GBs of RAM.

to 30 vertices, compared to 14 previously (the bottleneck here is the exponential number of hypothetical intervention results).

Hence, we argue that these methods are feasible in most practical settings as, on the one hand, sparse graphs are more prevalent and, on the other hand, this experiment considers fully undirected graphs and usually many edge directions are already detected during the estimation of the CPDAG. Finally, the tradeoff between possibly saved computation time and finding better intervention targets, should lean, in our view, towards the latter, as the cost of experimentation exceeds the cost of beforehand-computation by a large margin.

3.2 Estimating Causal Effects from CPDAGs and Observed Data

A second application concerns calculating the *total causal effects* to measure the effects of interventions in observational studies using Pearl's *do*-calculus (Pearl, 2009). For a given DAG $D = (V, E)$, with vertices $V = \{1, \dots, n\}$ representing random variables X_i, for $i \in V$, and a distribution f over $X = (X_1, \dots, X_n)$, which is Markov to D, the distribution generated by an intervention on X_i, written $do(X_i = x_i')$, can be expressed in a truncated factorization formula $f(x_1, \dots, x_n \mid do(X_i = x_i')) = \prod_{j=1, j\neq i}^{n} f(x_i \mid pa_i(D))|_{x_i = x_i'}$ if $x_i = x_i'$ and 0 otherwise. By integrating out all variables, except x_i, x_j, we get that the distribution of X_j after an intervention $do(X_i = x_i')$ can be computed by *adjustment for direct causes* of X_i (represented as parents of i in D):

$$f(x_j \mid do(X_i = x_i')) \;=\; \begin{cases} f(x_j) & \text{if } j \in Pa_i \\ \int f(x_j \mid x_i', pa_i) f(pa_i)\; d\, pa_i & \text{if } j \notin Pa_i, \end{cases} \tag{5}$$

where Pa_i stands for $Pa_i(D)$, for short, and $f(\cdot)$ and $f(\cdot \mid x_i', pa_i)$ represent preintervention distributions (Pearl, 2009, Theorem 3.2.2). The expected value

$$\mathbb{E}(X_j \mid do(X_i = x_i')) \;=\; \begin{cases} \mathbb{E}(X_j) & \text{if } j \in Pa_i \\ \int \mathbb{E}(X_j \mid x_i', pa_i) f(pa_i)\; d\, pa_i & \text{if } j \notin Pa_i, \end{cases}$$

which summarizes the intervention distribution, can be used to define the total causal effect of X_i on X_j as follows: For non-parametric models the formula

$$\theta_{ji}(D) \;=\; \mathbb{E}(X_j \mid do(X_i = x_i')) - \mathbb{E}(X_j \mid do(X_i = x_i'')) \tag{6}$$

can be taken as a definition of the effect, that expresses the difference between expected value of X_j under each of the following two actions: X_i is forced to take value x_i' or X_i is forced to take value x_i'' (Pearl, 2009, page 70); For parametric linear models, the causal effect can be defined as

$$\begin{aligned} \theta_{ji}(D) &= \tfrac{\partial}{\partial x}\mathbb{E}(X_j | do(X_i = x)) \\ &= \mathbb{E}(X_j \mid do(X_i = x_i' + 1)) - \mathbb{E}(X_j \mid do(X_i = x_i')) \end{aligned} \tag{7}$$

for any chosen value of x_i'. It shows the change in the expected value of X_j when changing in interventions the value of X_i by one unit.

Thus, when the true causal DAG D is known, the outcomes of interventions can be estimated using the formulas above and there is an extensive literature providing techniques

for calculating the causal effects when the formula (5) is not applicable, e.g., due to unobservable variables (see, e.g., (Pearl, 2009; Shpitser and Pearl, 2006; Shpitser et al., 2010; van der Zander et al., 2019)). On the other hand, if a DAG is only identifiable up to its MEC, the situation changes significantly: while for some CPDAGs one can compute the causal effects from the graph and the observed data (van der Zander and Liśkiewicz, 2016; Perković et al., 2017), in general, for a given CPDAG G, the true causal effect of X_i on X_j may differ across the DAGs in the MEC $[G]$. In such cases we can at best determine a *multiset* of possible causal effects $\theta_{ji}(D)$, one for each DAG D in $[G]$.

Based on this idea, Maathuis et al. (2009) propose algorithms, called IDA (Intervention Calculus when the DAG is Absent), to extract useful causal information, e.g., to estimate bounds on causal effects. The basic algorithm (Algorithm 1 in (Maathuis et al., 2009)), also called global-IDA in the literature, starts with an empty multiset Θ and, for a given CPDAG G, adds $\theta_{ji}(D)$ to Θ for all DAGs D in $[G]$. To avoid the unnecessary enumeration of all DAGs, the authors propose a natural modification, which computes the same output Θ and works as follows: Let $D_1, \dots, D_k \in [G]$ be DAGs such that $[D_1]_{\{\emptyset,\{i\}\}} \dot\cup \dots \dot\cup [D_k]_{\{\emptyset,\{i\}\}}$ is the partition of $[G]$ into the possible interventional-MECs for $\mathcal{I} = \{\emptyset, \{i\}\}$. Then, for all $\ell = 1, \dots, k$, the algorithm adds c_ℓ copies of $\theta_{ji}(D_\ell)$, where the multiplicity c_ℓ denotes the number of DAGs in $[D_\ell]_{\{\emptyset,\{i\}\}}$. The correctness is based on the fact that for all DAGs in $[D_\ell]_{\{\emptyset,\{i\}\}}$, the causal effect of X_i on X_j is the same.[13]

Maathuis et al. (2009) notice, that global-IDA *"works well if the number of covariates is small, say less than 10 or so"* and that the bottleneck is the computation of the multiplicities c_ℓ, which quickly becomes infeasible if the number of covariates n increases. Therefore, the authors developed a "localized" version, called local-IDA, which computes the multiset $\Theta^L = \{\theta_{ji}(D_1), \dots, \theta_{ji}(D_k)\}$ instead of Θ. This, however, does not reflect the true multiplicities assuming each DAG in the MEC is equally likely to be the ground truth. In the context of Fig. 3, assuming we perform IDA on CPDAG G, this means that the information, that the causal effect θ_{41} corresponding to the configuration $3 \to 1 \leftarrow 2$ appears in 6 of the possible 10 DAGs, is discarded.

In this paper we show that using our new approaches we can effectively implement global-IDA, meaning, in particular, we can efficiently compute for each possible parent set (i.e., each of the partitions of the MEC) the multiplicity, meaning the number of DAGs in the partition. We present this implementation as Algorithm 3. Relying on the formulation of the problem in terms of $\mathcal{I}$-MECs from above and utilizing Theorem 5 and 6, we can conclude:

Theorem 7 *For a given CPDAG $G = (V, E)$ and $i, j \in V$, Algorithm 3 computes the multiset $\Theta = \{\theta_{ji}(D) \mid D \in [G]\}$ including causal effects of X_i on X_j for all DAGs $D \in [G]$. It runs in time $\mathcal{O}(k \cdot p(|V|, |E|))$, where $k = |\Theta^L|$ and p is a polynomial bounding the time complexity of the Clique-Picking algorithm.*

Hence, the additional effort of Algorithm 3 compared to local-IDA is only a polynomial factor. In practice, this factor will likely not matter, as we show by replicating the experiments on linear models originally performed by Maathuis et al. (2009).

13. In the original work as well as later papers Ghassami et al. (2019) the connection to $\mathcal{I}$-MECs was not drawn. This is crucial for obtaining efficient algorithms for computing the multiplicities through Theorem 5 and connects the two applications discussed in this paper.

input : A CPDAG $G = (V, E)$ and vertices $i, j \in V$.
output: Multiset Θ of possible causal effects of X_i on X_j
$\Theta \leftarrow \emptyset$
$H \leftarrow$ undirected component of G containing vertex i
foreach *clique* $K \subseteq N_H(i)$ **do**
  for G, i, K compute $G' = \mathcal{E}_{\{\emptyset,\{i\}\}}(D_K)$ using Theorem 6
  $c \leftarrow \#\text{EXT}(G')$ using Algorithm 1
  add c copies of $\theta_{ji}(D_K)$ to Θ
end
return Θ

Algorithm 3: An efficient implementation of the global-IDA algorithm.

In a causal *linear model*, where every edge represents a linear direct causal effect and under the assumption that the distribution of variables is multivariate normal, one can compute the causal effects $\theta_{ji}(D)$, defined in Eq. (7), as the regression coefficient $\beta_{ji|Pa_i(D)}$ of X_i in the linear regression of X_j on X_i and $Pa_i(D)$ (for details, see e.g., (Maathuis et al., 2009)). Maathuis et al. (2009) use *sample versions* of global- and local-IDA, particularly relying on the PC-algorithm and conditional independence tests for the estimation of CPDAG G from data. In their studies, they consider variables $X_1, \ldots, X_n, X_{n+1}$ and, as described above, they compute the multisets $\hat{\Theta}_i$ and $\hat{\Theta}_i^L$ for total effects of a randomly chosen covariate X_i on a response variable $Y = X_n + 1$.

They report, that in simulation studies over sparse DAGs, already for $n = 14$, at least one of the 10 replicates[14] of the global-IDA algorithm took more than 48 hours to compute, so that the computation was aborted. Likewise, for the *riboflavin data* with $n = 4088$ covariates in the data set, the global-IDA algorithm is stated as infeasible. We perform the same experiments reporting (i) the run time of computing $\hat{\Theta}_i^L$, i.e., the causal effect for each possible parent set of X_i and (ii) the run time of computing the multiplicity for each parent set, i.e., the number of DAGs in the MEC with X_i having the specified parents, as proposed in Algorithm 3. Hence, local-IDA would be identical to performing step (i), whereas global-IDA would consist of (i) and (ii). Table 2 shows the results.

Clearly, the extra effort of computing the multiplicities is negligible in this setting. The regression tasks, which both local- and global-IDA perform for the causal effect estimation, have significantly larger computational effort. This is due to the fact that the counting tasks for computing the multiplicities is often *extremely simple* for the given graphs. The CPDAGs learned from the PC-algorithm are usually very sparse. Moreover, the input to Clique-Picking (and other counting algorithms) consists only of the undirected components of the CPDAG and even for large graphs, these are often quite small. In this sense, the graphs used in the experiments in the previous subsection were worst-case inputs when it comes to computational cost, as they were completely undirected. Therefore, we reemphasize that for most practical problems, there is no reason to avoid the counting task as the Clique-Picking algorithm should be fast enough to handle almost all imaginable cases.

14. In the original experiments, the algorithms run over 10 replicates with sample size 1000. We perform 100 replicates for more stable results.

Table 2: Mean runtime in seconds of computing the causal effects and the multiplicities over 100 replicates with sample size 1000 and the specified number of covariates. The case $n = 4088$ is the real-world riboflavin dataset and is hence performed only once. Here, we average over *all* covariates X_i instead of choosing a random one.

		number of covariates						
		4	9	14	29	49	99	4088
Effects	Time in s	0.09440	0.14986	0.18639	0.24875	0.24214	0.27393	0.48184
	Std. dev.	0.11958	0.11310	0.12029	0.16941	0.11084	0.14278	0.22053
Multipl.	Time in s	0.00014	0.00023	0.00023	0.00086	0.00113	0.00172	0.07042
	Std. dev.	0.00028	0.00059	0.00006	0.00453	0.00484	0.00545	0.02758

4. The Clique-Picking Algorithm

In this section, we will show how the problem #AMO defined on undirected chordal graphs can be solved in polynomial-time through the novel Clique-Picking algorithm. As discussed in Section 2, such a polynomial-time algorithm for #AMO means that the problem of computing the size of an MEC can be solved in polynomial-time as well.

The algorithm Clique-Picking, which we develop in the following, heavily relies on the special properties of chordal graphs and their close connection to AMOs. Hence, we start by introducing the necessary graphical terms and give important facts.

4.1 Further Definitions and Known Properties

Chordal graphs. The set of all maximal cliques of undirected graph G is denoted by $\Pi(G)$. A vertex is *simplicial* if its neighbors form a clique. 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[15]. 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.

AMOs. Just like any other DAG, an AMO α of a UCCG 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 $u \rightarrow v$ in α, u precedes v in τ. Note that every AMO of a UCCG contains exactly one source vertex, i.e., a vertex with no incoming edgesHe et al. (2015). Based on this observation, one may define the s-orientation G^s of a UCCG G to be 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

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

et al., 2015). This observation enables recursive strategies for counting AMOs: the "root-picking" approaches (He et al., 2015; Ghassami et al., 2019; Talvitie and Koivisto, 2019; Ganian et al., 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.

4.2 Basics of the Clique-Picking Algorithm

To count the number of Markov equivalent DAGs, we use the association between an AMO and its topological orderings. In accordance with the algorithm we develop, it is helpful to consider only topological orderings, which are well-behaved in the following sense:

Definition 8 *A topological ordering τ of an AMO α is called* clique-starting *if it has a maximal clique as a prefix.*

We denote all clique-starting topological orderings representing an AMO α of a graph G by $\text{top}_G(\alpha) = \{\tau_1, \ldots, \tau_\ell\}$ and will only consider such topological orderings in the following. It is sound to restrict ourselves in this way due to the following result:

Lemma 9 *Every AMO can be represented by a clique-starting topological ordering.*

Based on these observations, we generalize the definition of s-orientations with the goal of handling whole cliques at once: For this, we consider permutations π of a clique K, as each $\pi(K)$ represents a distinct AMO of the subgraph induced by K.

Definition 10 *Let G 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 be 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 components of $G^{\pi(K)}[V \setminus K]$ and let $\mathcal{C}_G(K)$ denote the undirected components of $G^K[V \setminus K]$.*

Figure 5 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. 5, 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 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 as:

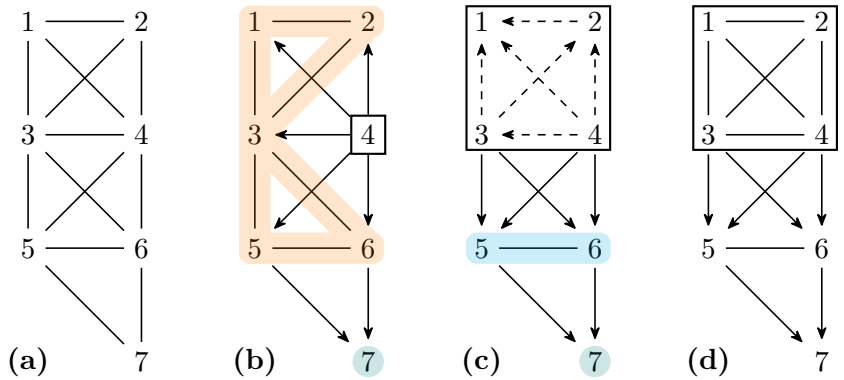


Figure 5: For a UCCG G in **(a)**, the figure shows $G^{(4)}$ in **(b)**, $G^{(4,3,2,1)}$ in **(c)**, and $G^{\{1,2,3,4\}}$ in **(d)**. The undirected components in $G^{(4)}$ and $G^{(4,3,2,1)}$ are indicated by the colored regions and the vertices put at the beginning of the topological ordering by 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 11 *Let G be a UCCG and K be a clique of G. For each permutation $\pi(K)$, all edges of $G^{\pi(K)}$ coincide with the edges of G^K, excluding the edges connecting the vertices in K, and therefore, in particular, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(K)$.*

In our journey towards a polynomial-time algorithm for #AMO, we start by developing a linear-time algorithm for computing $\mathcal{C}_G(K)$. This algorithm will yield structural insights we will later use for deriving a recursive formula for counting Markov equivalent DAGs.

The computation of $\mathcal{C}_G(K)$ can be performed efficiently through adaptions of well-known graph traversal algorithms used most prominently in chordality testing. While in (Wienöbst et al., 2021b) specifically the Lexicographic BFS algorithm has been used to compute $\mathcal{C}_G(K)$, we now propose a more general framework which allows "plugging in" various linear-time chordality testing algorithms.

To do this, we need to introduce further terms from the chordal graph theory and connect them to the problem of counting AMOs.

Definition 12 *A linear ordering $\rho = (x_1, \dots, x_n)$ of the vertices of chordal graph G is called a perfect elimination ordering (PEO) if for each $i \in \{1, \dots, n\}$ the vertex x_i is simplicial in $G[\{x_i, \dots, x_n\}]$.*

Lemma 13 *A topological ordering τ of the vertices of a UCCG G represents an AMO if, and only if, it is the reverse of a PEO.*

There are various linear-time graph traversal algorithms designed for chordality testing, such as the Lexicographic BFS, Maximum Cardinality Search, Lexicographic DFS and Maximal Neighborhood Search (Rose et al., 1976; Tarjan and Yannakakis, 1984; Corneil and Krueger, 2008). All of these algorithms are based on the same principle: the vertices are visited in reverse order of a PEO if, and only if, the graph is chordal. Hence, by Lemma 13 these algorithms will traverse the graph in the topological order of an AMO. A fact we exploit in the following.

To make our main algorithm independent on the specific method of graph traversal, we present Algorithm 4 for computing the set $\mathcal{C}_G(K)$ in a general form. It is based on a generic algorithm, called Maximum Label Search (MLS), to compute a PEO for a given graph G (Berry et al., 2009). MLS traverses G using the following labeling structure:

Definition 14 (Berry et al. (2009)) *A labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$ consists of:*

- *L is a set (the set of labels),*
- *$\preceq$ is a partial order on L (which may be total or not),*
- *l_0 is an element of L (the initial label),*
- *Inc (increase) is a mapping from $L \times \mathbb{N}^+$ to L satisfying the following IC (Inclusion Condition): for any subsets I and I' of $\mathbb{N}^+$, if $I \subset I'$, then $lab_{\mathcal{L}}(I) \prec lab_{\mathcal{L}}(I')$. Here, $\mathbb{N}^+$ denotes the set of positive natural numbers and for a set $J = \{i_1, i_2, \ldots, i_k\}$ with $i_1 > \cdots > i_k$, the labeling function is defined as $lab_{\mathcal{L}}(J) = Inc(\ldots(Inc(l_0, i_1), \ldots), i_k)$,*

Based on this, the MLS algorithm takes as input a graph G and for a specific labeling structure $\mathcal{L}$ it returns a PEO if G is chordal (see Algorithm MLS in (Berry et al., 2009)). In this framework, for example, Maximum Cardinality Search is a special case of MLS with labeling set $L = \mathbb{N}^+ \cup \{0\}$, the total order $\preceq$ to be $\leq$, $l_0 = 0$, and $\mathrm{Inc}(l, i) = l + 1$. Our adaption of MLS, which computes $\mathcal{C}_G(K)$, is presented as Algorithm 4.

input : A UCCG $G = (V, E)$, a clique $K \subseteq V$.
framework: A labeling structure $\mathcal{L} = (L, \preceq, l_0, Inc)$.
output : $\mathcal{C}_G(K)$.

$V' \leftarrow \emptyset$; $P \leftarrow \emptyset$; For all $x \in V$ initialize labels $L(x)$ as l_0
for $i = 1$ *to* n **do**
  if $i \leq |K|$ **then**
    $x \leftarrow$ any vertex in $K \setminus V'$
  else
    $X \leftarrow$ set of vertices in $V \setminus V'$ with maximal label
    Append undirected components of $G[X \setminus P]$ to the output
    $P \leftarrow P \cup X$
    $x \leftarrow$ any vertex in X
  end
  foreach y *in* $N(x) \setminus V'$ **do**
    $L(y) \leftarrow Inc(L(y), n - i + 1)$
  end
  $V' \leftarrow V' \cup \{x\}$
end

Algorithm 4: A generic algorithm for computing the set $\mathcal{C}_G(K)$.

For convenience, we introduce the following terms:

Definition 15 *In the execution of Algorithm 4 on input $G, \mathcal{L}$, let $P_i(y)$ be the set of previously visited neighbors of vertex $y \in G$ (a vertex is visited if it was chosen as x in line 4 or 9) before the start of the i-th iteration. Moreover, let $i(x)$ be the iteration in which vertex $x \in V \setminus K$ was output.*

Clearly, $P_i(y)$ are exactly the vertices, which contributed to y's label up to iteration i.

Lemma 16 *Algorithm 4 always chooses vertex x with maximal label.*

Proof This is the case by construction in line 9. We have to show that it also holds in line 4. Observe that the first $|K|$ chosen vertices are all from clique K. Hence, when a vertex x from K is chosen all previously chosen vertices are neighbors of x. This means that any other label is equal or smaller. ■

This lemma implies that Algorithm 4, as the Maximum Label Search, visits the vertices in reverse PEO order, i.e., in an order representing an AMO.

Theorem 17 *Algorithm 4 computes $\mathcal{C}_G(K)$. Moreover, it can be implemented to run in time $\mathcal{O}(|V| + |E|)$.*

Proof Consider two adjacent vertices a and b in G. We show that a and b are in the same subgraph in the output iff we have $a - b$ in G^K. By transitivity it follows that two vertices are in the same subgraph iff there is an undirected path between them, which implies the first part of the statement (we will analyze the run time afterwards).

If a and b are in the same connected component output by Algorithm 4 then there was a point in the algorithm at which a and b had a maximal label and, hence, either one could have been chosen as vertex x. In both cases the algorithm would have produced a topological ordering representing an AMO starting with clique K (following from Lemma 16 and Lemma 13), one time with $a \to b$, the other with $a \leftarrow b$. Hence, we have $a - b$ in $G^K[V \setminus K]$ by definition.

Consider that a and b are not in the same connected component output by the algorithm. Let a be w.l.o.g. the vertex which is output earlier, i.e., $i(a) < i(b)$. We show by induction over the order the vertices were visited that $a \to b$ in $G^K[V \setminus K]$. For the start of the induction, observe that the vertices in K are not output at all and all edges from K to vertices in $V \setminus K$ are oriented towards those vertices.

At the iteration $i = i(a)$ when a was output, b had a strictly smaller label. It follows that $P_i(a) \neq P_i(b)$. With $P_i(b) \setminus P_i(a) = \emptyset$ as the algorithm produces a PEO by Lemma 16, it follows $P_i(b) \subset P_i(a)$. Let c be in $P_i(a) \setminus P_i(b)$. By induction hypothesis, we have $c \to a$ in G^K as c is not output together with a (recall that i is the iteration when a is output, c has already been visited previously). Then, $a \to b$ follows from the first Meek rule.

Common choices of labeling structure (such as for Maximum Cardinality Search or Lexicographic BFS) lead to a linear-time implementation. ■

Theorem 17 is an important result in its own right. Algorithm 4 may be used not only for computing $\mathcal{C}_G(K)$, but also for computing the s-orientations of a chordal graph G in linear time as well as the interventional essential graph based on given intervention results (see Section 3 for a discussion).

But for now, we focus on the structural properties regarding AMOs and chordal graphs revealed by Algorithm 4, which allow us to conclude that the undirected components of G^K (i.e., the graph which occurs when fixing clique K as "source") are chordal and can be oriented independently. The first fact can be easily seen as, by Algorithm 4, the undirected components are induced subgraphs, which preserve the chordality of the graph. The second fact is more technical and due to the observation that vertices in the same undirected component have the same parent set in G^K, which ensures that any AMO of the component will not create a new v-structure in G^K. Crucially, this paves the way towards a recursive formulation of #AMO based on picking a clique as source.

Corollary 18 *Let G be a chordal graph and K a clique.*

1. *The undirected components of $G^K[V \setminus K]$ are induced subgraphs and hence chordal graphs.*
2. *Let adjacent x, y in G be in different undirected connected components of $G^K[V \setminus K]$ and $i(x) < i(y)$. Then, $x \to y$ is an edge in $G^K[V \setminus K]$.*
3. *$P_{i(v)}(v) = Pa_v(G^K)$.*
4. *For adjacent a, b in the same undirected component of $G^K[V \setminus K]$, we have that $P_{i(a)} = P_{i(b)}$.*
5. *The number $\#\mathrm{EXT}(G^{\pi(K)})$ can be factorized as*

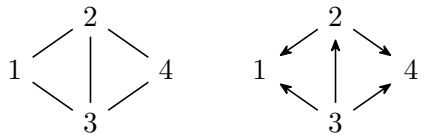
$$\#\mathrm{EXT}(G^{\pi(K)}) = \prod_{H \in \mathcal{C}_G(\pi(K))} \#\mathrm{AMO}(H).$$

6. *The number of AMOs represented by some topological ordering with clique K at the beginning (in any permutation) is*

$$|K|! \times \#\mathrm{EXT}(G^K) = |K|! \times \prod_{H \in \mathcal{C}_G(K)} \#\mathrm{AMO}(H).$$

In line with our notation, we write $\#\mathrm{EXT}(G^K)$ as this graph is partially oriented and $\#\mathrm{AMO}(H)$ with $H \in \mathcal{C}_G(K)$ as it is an undirected chordal graph. Based on item 6 of Corollary 18, we would like to count the AMOs of a chordal graph G with the following recursive procedure: Pick a maximal clique K, consider all its permutations at once (i.e., multiply by $|K|!$), and take the product of the recursively computed number of AMOs of the UCCGs of $\mathcal{C}_G(K)$. By Lemma 9, 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 starting with different maximal cliques. 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 2 *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 19 *Let α be an AMO of a chordal graph G and let $\tau_1, \tau_2 \in \mathrm{top}(\alpha)$ be two clique-starting 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, defined below, plays a crucial role to avoid overcounting.

Definition 20 *For a set S and a collection $\mathcal{R}$ of subsets of S, we define $\phi(S, \mathcal{R})$ as the number of all permutations of S that do not have a set $S' \in \mathcal{R}$ as prefix.*

Example 3 *Consider the set $S = \{2, 3, 4, 5\}$ and the collection $\mathcal{R} = \big\{\{2, 3\},\ \{2, 3, 5\}\big\}$. Then $\phi(S, \mathcal{R}) = 16$ 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 $\mathcal{R} \subseteq \Delta(G)$. Therefore, we can use the abbreviation $\phi(S) = \phi\big(S, \{\, S' \mid S' \in \Delta(G) \wedge S' \subsetneq S \,\}\big)$.

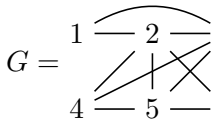
Proposition 21 *Let G be a UCCG. Then:*

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

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 19. 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 $\tilde{S} \in \Delta(G) \cup \Pi(G)$.

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 $\phi(\tilde{S})$. ■

Example 4 *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 21, 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 21 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 naïve enumeration. In general, however, this is a non-trivial task. We tackle this issue below.* ◇

4.3 The Algorithm

From Proposition 21 we know how to count AMOs by using minimal separators in order to avoid overcounting and it is rather easy to check that we can compute $\phi(S, \mathcal{R})$ in time *exponential* in $|\mathcal{R}|$ using the inclusion-exclusion principle. However, our goal is *polynomial time* and, thus, we have to restrict the collection $\mathcal{R}$.

Lemma 22 *Let* S *be a set and* $\mathcal{R} = \{X_1, \ldots, X_\ell\}$ *be a collection of subsets of* S *with* $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$*. Then:*

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

Proof We prove the statement by induction over ℓ with the base case $\phi(S, \emptyset) = |S|!$. Consider a set S and a collection $\mathcal{R} = \{X_1, \ldots, X_\ell\}$ of subsets of S. We can compute $\phi(S, \mathcal{R})$ 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.,

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

Inserting the induction hypothesis, we obtain:

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

■

Observe that this formula can be evaluated in polynomial time with respect to $|S|$ and ℓ, as all recursive calls have the form $\phi(X_i, \{X_1, \ldots, X_{i-1}\})$ and, thus, there are at most ℓ distinct ones. The goal of this section is to develop a version of Proposition 21 based on this lemma. This will allow us to obtain the Clique-Picking algorithm.

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 $\text{children}_T(v)$. 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 $S \subseteq V_G$ 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 23 *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 consist of all intersections $\iota(x_i) \cap \iota(x_{i+1})$ that are contained in $\iota(v)$, for $1 \leq i < p$.*

Lemma 24 *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 $\mathrm{FP}(v, \mathcal{T})$. ■

By combining the lemma with Lemma 22, 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. The formula given in proposition below achieves this goal.

Proposition 25 *Let G be a UCCG and $\mathcal{T} = (T, r, \iota)$ be a rooted clique tree of G. Then*

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

Crucially, evaluating this formula can be done efficiently. As implementation, we give Algorithm 5, which utilizes memoization to avoid recomputations. Traversing the clique tree with a BFS allows for a straightforward computation of FP.

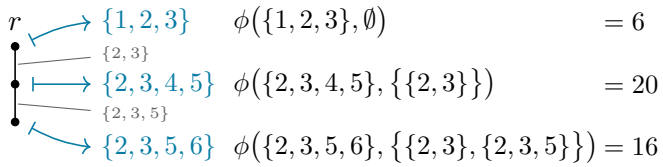
input : A UCCG $G = (V, E)$.
output: #AMO(G).
return `count`$(G, \emptyset)$
function `count`(G, memo)
  if $G \in$ memo **then**
    return memo$[G]$
  end
  $\mathcal{T} = (T, r, \iota) \leftarrow$ a rooted clique tree of G
  sum $\leftarrow 0$
  $Q \leftarrow$ queue with single element r
  while Q *is not empty* **do**
    $v \leftarrow$ `pop`(Q)
    `push`$(Q, \text{children}(v))$
    prod $\leftarrow 1$
    foreach $H \in \mathcal{C}_G(\iota(v))$ **do**
      prod $\leftarrow$ prod $\cdot$ `count`(H, memo)
    end
    sum $\leftarrow$ sum $+ \phi(\iota(v), \text{FP}(v, \mathcal{T})) \cdot$ prod
  end
  memo$[G]$ = sum
  return sum
end

Algorithm 5: Clique-Picking

Theorem 26 *For an input UCCG G, Algorithm 5 returns the number of AMOs of G.*

We defer the rather involved proof of this Theorem to Section 8.

Example 5 *We consider a rooted clique tree (T, r, ι) for the graph G from Example 4. 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 5 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:

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

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 22 and 24. We prove next that Algorithm 5 performs at most $2 \cdot |\Pi(G)| - 1$ recursive calls, which implies overall polynomial run time.

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

We can, hence, conclude the following:

Theorem 28 *The Clique-Picking algorithm runs in time* $\mathcal{O}\big(\, |\Pi(G)|^2 \cdot (|V| + |E|) \,\big)$.

Proof By Proposition 27, `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 4 in time $\mathcal{O}(|V| + |E|)$ by Theorem 17.

For the computation of $\phi(S, \mathrm{FP}(v, \mathcal{T}))$, note that FP can be obtained 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 using dynamic programming and the recursive formula from Lemma 22. There are $\mathcal{O}(|S|)$ distinct recursive calls 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 $\mathcal{O}(|E|)$. ■

We summarize the findings of this section in the following theorem which restates our main Theorem 1.

Theorem 29 *Algorithm 1 solves the problem* #EXT *for CPDAGs (i.e., the computation of the size of an MEC) in polynomial time.*

Proof The correctness of Algorithm 1 follows from Equation (1) and Theorem 26. By Theorem 28, it performs polynomially many arithmetic operations. Since $\#\mathrm{AMO}(G)$ is bounded above by $n!$, with n denoting $|V_G|$, all operations run in polynomial time because the involved numbers can be represented by polynomially many bits. ■

5. Uniform Sampling of Markov Equivalent DAGs

In this section, we investigate the problem of uniformly sampling a DAG from a Markov equivalence class. This problem is closely related to the counting problem and we show how to solve it efficiently using the Clique-Picking approach.

The general approach can be seen in Algorithm 6. The recursive function `sample` takes as input a UCCG G and produces a topological ordering of the vertices τ, which represents

input : A UCCG G.
output: Topological ordering of uniformly drawn AMO of G.

function `sample`(G)
  $\mathcal{T} = (T, r, \iota) \leftarrow$ rooted clique tree of G
  $v \leftarrow$ drawn with probability proportional to $\phi(\iota(v), \mathrm{FP}(v, \mathcal{T})) \times \prod_{H \in \mathcal{C}_G(\iota(v))} \#\mathrm{AMO}(H)$
  $\tau \leftarrow$ uniformly drawn permutation of $\iota(v)$ without prefix in $\mathrm{FP}(v, \mathcal{T})$
  foreach $H \in \mathcal{C}_G(K)$ **do**
    $\tau \leftarrow$ `concat`$(\tau,$ `sample`$(H))$
  end
  return τ
end

Algorithm 6: The recursive function `sample` uniformly samples an AMO (represented through its topological ordering) from a UCCG G.

a uniformly sampled AMO of G. It utilizes the formula

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

derived in Proposition 25. Hence, the counting is done with respect to a clique tree $\mathcal{T}$ of G.

The idea is to first sample a clique (i.e., a node v of the clique tree), which is put at the start of the topological ordering. For this, node v is drawn with probability proportional to

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

i.e., the number of AMOs counted at the clique. This will ensure that every AMO has uniform probability of being drawn. In practice, it is useful to run the Clique-Picking algorithm once as precomputation step, in order not to evaluate the formula repeatedly. We discuss such implementation details later. Next, a permutation τ of chosen clique K is drawn uniformly from those which do not start with one of the "forbidden" prefixes in $\mathrm{FP}(v, \mathcal{T})$. Recall that function ϕ counts only such permutations. Finally, the algorithm recurs, as prescribed by the formula above, into the subgraphs in $\mathcal{C}_G(K)$, which are considered independently. The topological orderings sampled for these subgraphs are appended to τ.

We will start this section by showing that this approach will indeed sample a uniform AMO. Afterwards, we will discuss possible implementations of this method.

Theorem 30 *For a UCCG G, the function* `sample` *returns a topological ordering representing an AMO chosen with uniform probability.*

Proof We show the theorem by induction. As base case we consider a single clique K. Here, any permutation of K represents a unique AMO. Because there is only one v to choose and, as $\mathrm{FP}(v, \mathcal{T})$ is empty, such a permutation (and hence the corresponding AMO) is chosen uniformly.

In order to make the following arguments more precise, we denote with $\Pr(\tau_\alpha(G))$ the probability that Algorithm 6 draws a topological ordering τ of the vertices in G that represents α. Our goal is to show, as we just did in the base case, that for all α:

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

For UCCG G and clique-tree $\mathcal{T}$, let v_α be the node in the clique-tree, at which α is "counted" and let π_α be the corresponding permutation of the clique $\iota(v)$ in any topological ordering of α. The correctness of the proof relies on the fact that both v_α and π_α are unique (for v_α this follows from the proof of Proposition 25). Then:

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

In the second step, we insert the definitions of $\Pr(v_\alpha)$ and $\Pr(\pi_\alpha \mid v_\alpha)$. In the third step, we use the induction hypothesis

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

to complete the proof. ■

We will now discuss how to efficiently implement the proposed sampling algorithm. The non-trivial tasks are lines 3 and 4 of Algorithm 6.

Note that when calling the function `sample` for an input graph G, it is only necessary to know for each node v in a certain clique-tree $\mathcal{T}$, the following information: the set $\mathrm{FP}(v, \mathcal{T})$ and

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

Moreover, in a single run of the counting algorithm (Algorithm 5) these terms are computed for G and all possible recursive subcalls. Hence, in a preprocessing step we perform the counting algorithm once, storing these information.

For the implementation of line 3, we hence need to draw from a categorical distribution with known weights over the nodes of clique-tree $\mathcal{T}$. This is possible in constant time $\mathcal{O}(1)$ using the Alias Method (Walker, 1974; Vose, 1991) assuming that the preprocessing includes the computation of a Alias Table. As this is possible in linear-time in the number of categories, there is no computational overhead.

The implementation of line 4 is trickier. In (Wienöbst et al., 2021b), we proposed a routine which performs this step in $\mathcal{O}(|\iota(v)|^2)$ time. This leads to overall cost of $\mathcal{O}(|V|+|E|)$ of `sample`. Moreover, the precomputation is significantly more complicated, needing time $\mathcal{O}(|\Pi(G)|^2 \cdot |V| \cdot (|V| + |E|)$ and hence an additional factor $|V|$.

Here, we propose a simple Monte Carlo algorithm for the implementation of line 4 based on rejection sampling. Due to the combinatorial structure of the counting function ϕ, we are able to bound the expected number of draws in this rejection sampling routine by a constant. This leads to a very efficient and practical algorithm, as the preprocessing cost are in the same order as the standard Clique-Picking algorithm, i.e., time $\mathcal{O}(|\Pi(G)|^2 \cdot (|V|+|E|))$, and the sampling of the topological ordering is even possible in time $\mathcal{O}(|V|)$.

Theorem 31 *There is an algorithm that, given a connected chordal graph G, uniformly samples a topological ordering of an AMO of G in* expected time $\mathcal{O}(|V|)$ *after an initial* $\mathcal{O}(\Pi(G)^2 \cdot (|V| + |E|))$ *setup.*

Theorem 2 (announced in the introduction) follows directly from the theorem above since to uniformly sample a DAG in MEC represented by a CPDAG C one can uniformly sample a topological ordering of an AMO of G, independently for each undirected component G of C and then combine the orderings to obtain a resulting DAG.

Proof (of Theorem 31) As discussed above, we implement line 4 in Algorithm 6 by rejection sampling, i.e., repeatedly draw random permutations until one which is not forbidden is found.

We begin by showing that, in expectation, only a constant number of draws are necessary (this holds for any input). Let $\phi(S, \mathrm{FP})$ be the number of allowed permutations. The ratio $\frac{\phi}{|S|!}$ gives the probability that a random permutation is allowed. We have to find a lower bound for the ratio in order to obtain the statement. Given a set S, the value of ϕ reaches its minimum when allowing as few prefixes as possible. Consequently, a worst-case collection for $S = \{s_1, ..., s_p\}$ is $\mathrm{FP} = \{\{s_1\}, \{s_1, s_2\}, ..., \{s_1, s_2, ..., s_{p-1}\}\}$.

In this case, the number of allowed permutations is known as the number of irreducible permutations (OEIS A003319 (OEIS Foundation Inc., 2022)), which we denote with $\rho(p)$. It is well-known (and a special case of Lemma 22):

$$\rho(p) = p! - \sum_{i=1}^{p-1} i! \cdot \rho(p-i).$$

For our derivation of the lower bound, we start by deriving some simple bounds of fractions of binomial coefficients. For $2 \leq i \leq p-2$

$$\frac{1}{\binom{p}{i}} \leq \frac{1}{\binom{p}{2}} = \frac{2}{p(p-1)}$$

holds and therefore

$$\sum_{i=1}^{p-1} \frac{1}{\binom{p}{i}} = \frac{2}{p} + \sum_{i=2}^{p-2} \frac{1}{\binom{p}{i}} \leq \frac{2}{p} + \frac{2(p-3)}{p(p-1)} \leq \frac{4}{p}.$$

Computing the ratio and using the inputs S, FP as defined above, we have for $p \geq 8$

$$\begin{aligned}\frac{\phi(S,\mathrm{FP})}{|S|!} \geq \frac{\rho(p)}{p!} &= 1 - \sum_{i=1}^{p-1} i! \cdot \frac{\rho(p-i)}{p!} \\ &\geq 1 - \sum_{i=1}^{p-1} i! \cdot \frac{(p-i)!}{p!} = 1 - \sum_{i=1}^{p-1} \frac{1}{\binom{p}{i}} \\ &\geq 1 - \frac{4}{p} \geq 1 - \frac{1}{2}.\end{aligned}$$

Hence,

$$\frac{\phi(S,\mathrm{FP})}{|S|!} \geq \frac{1}{2}$$

for $|S| \geq 8$; that the estimate holds for all $|S| < 8$ can be checked by hand. In conclusion, it holds

$$\mathbb{E}[\text{number of trials until first success}] \leq \frac{1}{\frac{1}{2}} = 2.$$

It remains to analyze the expected run time of this routine. Drawing a permutation is possible in linear time in $|S|$. Note that FP can be efficiently represented by only storing the new elements of X_i (recall that $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_p$). Checking whether a permutation is forbidden can be done in linear-time as well: For every object $s \in S$, we record its first occurrence in FP. If it first occurred in set X_k, we have $o[s] = \sum_{i=1}^{k} |X_i|$; otherwise, it is in no set of forbidden prefixes and we put $o[s] = p + 1$. Afterwards, we go through the drawn permutation from front to back and memorize the highest o-value seen up until this step. If at position i the maximal value has been i, we can conclude that this permutation contains a forbidden prefix.

We will now discuss the run time of the whole `sample` function: We assume that as precomputation, a modified version of the Clique-Picking was performed. Then, using the Alias Method, line 3 takes time $\mathcal{O}(1)$.

Hence, we have overall expected linear-time for the drawing of a non-forbidden permutation. This means, we "pay" a constant amount per element in the build topological order and therefore this order can even be obtained in expected time $\mathcal{O}(|V|)$ after appropriate preprocessing. Note that to output the AMO itself, $\Theta(|V| + |E|)$ time is needed as this is the size of the output, but in a lot of cases the topological ordering might be sufficient. ■

We close this section by giving an experimental evaluation of our algorithm. As there are, to the best of our knowledge, no other implementations of exact sampling from an MEC, we will confine ourselves to showing that (i) the overhead of the preprocessing for sampling compared to the "standard" Clique-Picking algorithm is negligible and (ii) that sampling after preprocessing is extremely fast. We compare implementations of the algorithms in Julia and generated chordal graphs as described in (Wienöbst et al., 2021b), namely using the subtree intersection method (Seker et al., 2017) with density parameter $k = \log n$ (the expected number of neighbors per vertex is proportional to this parameter) and the algorithm by Scheinerman (1988) for sampling random interval graphs (interval graphs form a subclass of chordal graphs). For each input graph, we performed the counting algorithm without and with preprocessing. The run times are averages over 100 graphs. Afterwards,

we sampled 10 DAGs from each MEC uniformly, in total forming the average over 1000 sampling steps.

Table 3: The run times in seconds of the standard Clique-Picking algorithm without any precomputations (CP w/o pre.) and the modified one which includes precomputations (CP with pre.) for sampling on randomly generated chordal graphs (using the subtree intersection method as well as random interval graphs). For each choice of parameters, the algorithms were run on the same 100 graphs. Moreover, we give the average run time of sampling (after the preprocessing step), which is calculated as the average of 10 samplings per graph.

	Number of vertices								
	16	32	64	128	256	512	1024	2048	4096
Random subtree intersection ($k = \log_2 n$)									
CP w/o pre.	0.00076	0.00199	0.00729	0.02718	0.09463	0.38164	1.62875	7.53509	35.0380
CP with pre.	0.00135	0.00219	0.00774	0.02783	0.09602	0.38530	1.63844	7.58248	35.0759
Sampling	0.00001	0.00003	0.00006	0.00013	0.00026	0.00054	0.00118	0.00283	0.00695
Random interval graphs									
CP w/o pre.	0.00066	0.00211	0.00834	0.03512	0.18089	1.14654	8.17442	66.3541	539.270
CP with pre.	0.00080	0.00233	0.00864	0.03600	0.18278	1.15313	8.20020	66.2455	538.496
Sampling	0.00002	0.00003	0.00008	0.00025	0.00068	0.00204	0.00691	0.02298	0.10378

First, the run time difference between the standard Clique-Picking algorithm and the modified one, which includes preprocessing for sampling, is extremely small. The additional computations do not form the bottleneck of the approach and have only a small influence on the run time. For the very large graphs, in particular the dense interval graphs, the run time difference can hardly be measured, due to the fact that the additional precomputation effort is independent of the number of edges, which dominates the run time.[16]

Second, it can be clearly seen that sampling (after the initial setup step) is extremely fast. Even for large graphs it takes only fractions of a second. We remark that the sampling algorithm returned the full sampled DAG, which is the desired output in most cases, but that it would also be possible to only return the topological ordering, reducing the run time further.

6. Complexity of Counting Under Background Knowledge

As a generalization of the counting problems for MECs, we consider the problem of counting the number of DAGs in case of additional background knowledge. The formulation of the problem will not be different than before, we still want to compute #EXT(G) for a graph G, only now we do not make the assumption that G is a CPDAG (or interventional essential graph), but instead allow for arbitrary input graphs. This includes two well-known graph classes, the one of PDAGs and MPDAGs. A PDAG is a partially directed graph without

16. The execution time naturally fluctuates and for the large interval graphs this fluctuation influences the result more than the actual overhead. Hence, in some cases the precomputation algorithm is recorded as faster in the experiments. Clearly, Clique-Picking with precomputations does strictly more computations and, thus would, without noise, not be faster than normal Clique-Picking.

a directed cycle and an MPDAG is a PDAG, which has been maximally oriented using the Meek rules (Meek, 1995).

The following theorem shows that Theorem 29 is tight in the sense that counting Markov equivalent DAGs on the more general input graphs, which encode additional background knowledge (i.e., PDAGs or MPDAGs) is not in P under standard complexity-theoretic assumptions. We do this by reduction from the #P-hard problem of counting the number of topological orderings of a DAG (Brightwell and Winkler, 1991), in the following denoted by #TO.

Theorem 32 *The problem* #EXT *is #P-complete for arbitrary input graphs G, and in particular for PDAGs and MPDAGs.*

Proof We give a parsimonious reduction which by construction will consist of acyclic graphs, hence the hardness follows for PDAGs. The resulting PDAGs can moreover be transformed into an equivalent MPDAG (regarding the corresponding extensions) in polynomial time (Meek, 1995).

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.

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 extension 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'. ■

Notably, the reason Clique-Picking cannot be used to solve these counting problems can be directly connected to the main idea of the proof as well. Intuitively, the problems for PDAGs and MPDAGs 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 hard problem of counting the number of topological orderings of a DAG. We formalize this in the following. First, we introduce a modified version of counting function ϕ.

Definition 33 *For a set S, a collection $\mathcal{R}$ of subsets of S and a partial order $\preceq$ over the elements of S, we define $\phi'(S, \mathcal{R}, \preceq)$ as the number of all permutations of S consistent with $\preceq$ that do not have a set $S' \in \mathcal{R}$ as prefix.*

Hence, we generalize the function ϕ used in the Clique-Picking algorithm to counting only linear orderings (i.e., permutations) consistent with a given partial order. If $\mathcal{R}$ is empty, it coincides with the problem is of counting the extensions of a partial order. As this is equivalent to the problem #TO (all relations can be encoded as directed edges), we will denote by $\#\mathrm{TO}(S, \preceq)$ the number of linear orderings of S consistent with $\preceq$.

Lemma 34 *Let S be a set and $\mathcal{R} = \{X_1, \ldots, X_\ell\}$ be a collection of subsets of S with $X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_\ell$. Then, function $\phi'(S, \mathcal{R}, \preceq)$ can be computed by $\mathcal{O}(\ell^2)$ calls to #TO.*

Proof We base our approach on the recursive formula derived in Lemma 22

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

Instead of $|S|!$, compute the number of permutations of S consistent with $\preceq$. In the sum, check whether the partition in X_i (at the beginning of the permutation) and $S \setminus X_i$ (at the end of the permutation) violates the partial ordering (let indicator function $I(X_i, \preceq)$ denote this and evaluate to 0 if $\preceq$ is violated, else to 1). Replace $|S \setminus X_i|$ by the number of permutations of this subset of S which conforms to $\preceq$. We obtain:

$$\phi'(S, \mathcal{R}, \preceq) = \#\mathrm{TO}(S, \preceq) - \sum_{i=1}^{\ell} I(X_i, \preceq) \cdot \#\mathrm{TO}(S \setminus X_i, \preceq) \cdot \phi'(X_i, \{X_1, \ldots, X_{i-1}\}, \preceq).$$

Correctness follows as in Lemma 22 and as there are at most ℓ recursive calls, we have $\mathcal{O}(\ell^2)$ calls to #TO. ■

Theorem 35 *Counting the number of AMOs can be solved in time $\mathcal{O}(n^4 \cdot T(n))$ for PDAGs and MPDAGs, where $T(n)$ is the time required to solve an instance of #TO.*

Proof We consider the following algorithm (input is a PDAG or an MPDAG G)

1. Compute the CPDAG C, which contains all the DAGs represented by G (Wienöbst et al., 2021a). Note that a PDAG or an MPDAG represents a subset of an MEC, C is the CPDAG of this class.
2. Consider the UCCGs of C, compute the number of AMOs consistent with the edges in G for each, and multiply them. This way the number of AMOs of G can be obtained.

   We do the computation for each UCCG by calling a modified version of `count` from Algorithm 5 with additional parameter $\preceq$ and ϕ replaced by ϕ'. We pass this function a UCCG of C and as $\preceq$ we choose $\preceq_G$, i.e., the partial ordering over the UCCG given by the directed edges of G (i.e., $u \preceq_G v$ if $u \to v$ in G).

The correctness follows immediately, as Algorithm 5 considers every AMO once and this modification prunes exactly those AMOs not conforming to the background knowledge.

As `count` is called at most n times and there are at most n maximal cliques, the function ϕ' will be called at most n^2 times. In the worst case, evaluating ϕ' needs $\mathcal{O}(n^2)$ calls to #TO, thus we obtain the overall bound of $\mathcal{O}(n^4)$ calls. ■

In practice, the bound of $\mathcal{O}(n^4)$ oracle calls should be rather pessimistic as the parameter l in the computation ϕ', i.e., the number of forbidden prefixes, is usually rather small.

7. Conclusion

We presented the first polynomial-time algorithms for counting and sampling Markov equivalent DAGs. Crucially, our novel Clique-Picking approach is also extremely fast in practice. This means that especially the task of computing the size of an MEC does *not* have to be avoided, as we have argued by demonstrating the feasibility in two important applications. This enables researchers to choose more reliable and robust algorithms.

For the uniform sampling problem, we gave a new and simple linear-time algorithm after preprocessing with minimal overhead, which performs very well in practice, in particular, when many DAGs are sampled from the same MEC. Finally, we completed the theoretical study of the problem by showing that the more general problem with additional background knowledge is not solvable in polynomial-time under common complexity-theoretical assumptions, while also giving a reduction to classical counting problems.

8. Missing Proofs

8.1 Proof of Theorem 6 in Section 3

Proof A natural approach to compute G' for possible interventional values represented by K, is as follows. We orient the edges in G according to K and next apply directly the Meek rules (Meek, 1995). It has been shown that it is sufficient to only apply the first two Meek rules and with an efficient implementation utilizing the special structure of the problem this yields time $\mathcal{O}(d \cdot m)$ (AhmadiTeshnizi et al., 2020) (where d is the maximal degree of the graph). Below we show that using our methods we can compute G' in linear time $\mathcal{O}(n+m)$.

Due to Proposition 4 we know that, to compute G', it is sufficient to orient only H into H' since the remaining UCCGs of G remain unchanged. Let D be the set of vertices reachable from v (including v itself) in H with edges incident to K removed. Let $A = V \setminus \{D \cup K\}$ be the remaining vertices without K. As we will show in the following, (i) the induced subgraph $H[A \cup K]$ is undirected, (ii) there are no edges between A and D, (iii) the edges from K to D are oriented outwards from K (iv) and the edges in $H[D]$ are given by calling Algorithm 4 on $H[D \cup K]$ with clique $K \cup \{v\}$.

We begin with (ii). Assume, for the sake of contradiction there is an edge $A \ni a - v \in V$. Then, by definition, a would be part of V.

For (iii), observe that there is a path in $H[D]$ from v to every vertex. For the sake of the argument, let us only consider shortest paths. Then, the first Meek rule can be iteratively applied along that path (note that the first edge is given by the intervention result). Hence, in H', there is a directed path from v to any vertex in $H'[D]$. Consequently, every edge between K and D has to be oriented from K to D to avoid a directed cycle (every vertex in K is a parent of v in H').

We are now able to show (i). From (ii) and (iii), every edge between $A \cup K$ and D is oriented from $A \cup K$ to D. It follows that the chordal induced subgraph $H[A \cup K]$ can be oriented independently of the remaining graph as no v-structure nor cycle can occur.

It is left to show (iv). By the intervention result and (iii), we know that every edge from the initial clique $K \cup \{v\}$ is oriented outwards. It immediately follows from the correctness of Algorithm 4 that every implied directed edge is correctly detected (as it follows from those initial orientations). To see that all undirected edges $a - b$ are indeed undirected in

H', recall that in the proof of Theorem 17 it is argued that there exists an AMO with $a \rightarrow b$ and one with $a \leftarrow b$. Now note that finding an AMO for $H[D \cup K]$ (the orientation of the initial clique does not matter, just consider an arbitrary fixed orientation), will also yield an AMO for H by combining it with an AMO for $H[A \cup K]$. Hence, the same argument holds. ■

8.2 Missing Proofs in Section 4

8.2.1 Proof of Lemma 9

Proof Consider AMO α. We construct one-by-one a topological ordering starting with a maximal clique by an adaption of Kahn's algorithm (Kahn, 1962). First, let the start vertex in the ordering be the unique source s (recall that an AMO has a unique source vertex) and let set $S = \{s\}$ denote the already considered vertices. Second, as long as there is a vertex adjacent to every $x \in S$, choose such a vertex v which is incident to no edge $u \rightarrow v$ in α for $u \in V \setminus S$ and add it to S. Third, iteratively append the remaining vertices to the ordering by repeatedly choosing vertices with no incoming edges from unvisited vertices.

Clearly, the resulting ordering is a topological ordering and starts with a maximal clique provided vertex v always exists. Consider the set $W = \{w \mid w \in N(u) \text{ for all } u \in S\}$ of common neighbors of S, which is non-empty in the second phase. Assume for a contradiction that every vertex in W has an incoming edge from a vertex in $V \setminus S$. Note that no vertex in $w \in W$ can have an incoming edge from $x \in (V \setminus S) \setminus W$ as this would imply a v-structure $y \rightarrow w \leftarrow x$ for a $y \in S$ not adjacent to x in the given graph α. As the graph α is acyclic (and this property holds for taking induced subgraphs, i.e., for $G[W]$ as well) there has to be a vertex in W with no incoming edge – a contradiction. ■

8.2.2 Proof of Lemma 11

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) $a \rightarrow b \leftarrow c$. 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 $b \rightarrow c \notin K$ as c is not in K and thus preceded by b in τ', or
2. all three vertices are in K, but then $a \rightarrow b \leftarrow c$ is no induced subgraph as K is a clique.

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, $\mathcal{C}_G(\pi(K)) = \mathcal{C}_G(K)$. ■

8.2.3 Proof of Lemma 13

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. ■

8.2.4 Proof of Corollary 18

Proof

1. Follows immediately from Theorem 17.

2. Shown in the proof of Theorem 17.

3. We show two directions: Let $x \in P_{i(v)}(v)$. Then, x is a neighbor of v and output before v. By 2. we have $x \to v$. Now, let $x \in Pa_v(G^K)$, i.e., x is connected by a directed edge to v in G^K. From 1. it follows that x is not in the same connected component. Then x is visited before v is output and consequently in $P_{i(v)}(v)$.

4. As a and b are output in the same iteration, they both have the maximum label, and could both have been picked as vertex x. However, if $P_{i(a)}(a) \setminus P_{i(b)}(b) \neq \emptyset$ or $P_{i(b)}(b) \setminus P_{i(a)}(a) \neq \emptyset$ the algorithm would not produce the reverse of a PEO (after the choice of either a or b). A contradiction. Hence, the statement follows.

5. By 1. the undirected components are chordal induced subgraphs and hence its consistent extensions are AMOs. It is left to show that the orientations of the connected components can be constructed separately, yielding the product formula.By combining 3. and 4., the set of parents is identical for each vertex in the same component. Then, the statement follows from this fact analogously to Theorem 4 and 5 from Lemma 10 in (He and Geng, 2008).

6. By Lemma 11 we have that $\#\mathrm{AMO}(G^{\pi(K)}) = \#\mathrm{AMO}(G^K)$ for any permutation π. As there are $|K|!$ many permutations, which all lead to different AMOs, and combined with 5. we arrive at the stated formula. ■

8.2.5 Proof of Lemma 19

Proof Assume by Lemma 9 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 $u \rightarrow v$ 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 $P_1 = K_1 \backslash S$ from $P_2 = K_2 \backslash S$. Note that the minimality follows by definition. It remains to show that S indeed separates P_1 and P_2. For a contradiction, let $P_1 \ni x_1 - x_2 - \cdots - x_{k-1} - x_k \in P_2$ 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 $x_1 \rightarrow x_2$ 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 $x_{k-1} \rightarrow x_k$. However, τ_2 would imply the edge $x_{k-1} \leftarrow x_k$ in α – a contradiction. ■

8.2.6 Proof of Proposition 25

To ensure the property that we count every AMO *exactly* once, 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 (the one given in Proposition 25) 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 36 *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 6 *The* $\{2,3\}$*-flowers of the graph from Example 4 are* $\{\{1,2,3\}\}$ *and* $\{\{2,3,4,5\}, \{2,3,5,6\}\}$. ◇

Lemma 37 *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. ■

Lemma 38 *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 $G[V \setminus S]$. ■

Since for a $S \in \Delta(G)$ the subtree $\iota^{-1}(S)$ of (T, r, ι) is connected, Lemma 37 and Lemma 38 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 39 *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 37 and 38, 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. ■

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 40 *Let α be an AMO such that every clique-starting 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 $\tau \in \text{top}(\alpha)$ starting with K.*

Proof Let τ be a topological ordering representing α that starts with S. By Lemma 9, 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. ■

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 $\tau \in \text{top}(\alpha)$, 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) $F_1 \prec_T F_2$.

Now, we are ready to give:

Proof (of Proposition 25) We have to show that every AMO α is counted exactly once. Recall that $\text{top}(\alpha) = \{\tau_1, \ldots, \tau_\ell\}$ is the set of clique-starting 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 $\tau \in \text{top}(\alpha)$.

Claim 41 *There is a unique least maximal clique $K \in \Pi(G)$ with respect to $\prec_\alpha$.*

Proof Let $\text{top}'(\alpha) \subseteq \text{top}(\alpha)$ be an arbitrary subset of the clique-starting topological orderings that represent α and let μ be the number of different maximal cliques with which elements in $\text{top}'(\alpha)$ start. We prove the claim by induction over μ. In the base case, all elements in $\text{top}'(\alpha)$ 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 19, all $\tau \in \text{top}'(\alpha)$ start with the same $S \in \Delta(G)$.

Consider the bouquet $\mathcal{B}(S)$, which is partially ordered by $\prec_T$. Lemma 39 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 $\text{top}'(\alpha)$ to the set $\text{top}''(\alpha)$ of topological orderings that start with a maximal clique in F. This set is non-empty by Lemma 40 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 $S \in \text{FP}(\iota^{-1}(K), \mathcal{T})$ preventing α from being counted), and (ii) that α is not counted somewhere else (i.e., there is some set $S \in \text{FP}(\iota^{-1}(K'), \mathcal{T})$ for all other $K' \in \Pi(G)$ that can be at the beginning of some $\tau \in \text{top}(\alpha)$).

Claim 42 *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 $\tau \in \text{top}(\alpha)$. Then there is no $S \in \Delta(G)$ with $S \in \text{FP}(\iota^{-1}(K), \mathcal{T})$ 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 $S \in \text{FP}(\iota^{-1}(K), \mathcal{T})$, there is another flower $F' \in \mathcal{B}(S)$ with $F' \prec_T F$. Lemma 40 tells us that there is another clique $K' \in F'$ that is at the beginning of some $\tau' \in \text{top}(\alpha)$. However, then we have $K' \prec_\alpha K$ – contradicting the minimality of K. ■

Claim 43 *Let $\tau_1, \tau_2 \in \text{top}(\alpha)$ be two topological orderings starting with $K_1, K_2 \in \Pi(G)$, respectively. If $K_1 \prec_\alpha K_2$ then $K_1 \cap K_2 = S \in \text{FP}(\iota^{-1}(K_2), \mathcal{T})$.*

Proof Since K_1 and K_2 correspond to $\tau_1, \tau_2 \in \text{top}(\alpha)$, we have $K_1 \cap K_2 = S \in \Pi(G) \cup \Delta(G)$ by Lemma 19 – 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 by the definition of S-flowers and Lemma 38, 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, $S \in \text{FP}(\iota^{-1}(K_2), \mathcal{T})$. ■

This completes the proof of Proposition 25. ■

8.2.7 Proof of Theorem 26

Proof Observe that recursive calls are performed in line 14 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 25, as it traverses the clique tree with a BFS in order to compute the sets $\mathrm{FP}(v, \mathcal{T})$ and evaluates this formula. ■

8.2.8 Proof of Proposition 27

Let $V'_{i(x)}$ be the set of all visited vertices by Algorithm 4 in the step before x is output (i.e., the set V' at this point). Also recall the definition of $P_{i(x)}(x)$ as the already visited neighbors of x at the iteration when x is output. As all other vertices in the same component in $\mathcal{C}_G(K)$ as x have the same preceding neighbors, we will define $P_{i(H)}(H) := P_{i(x)}(x)$ for all $H \in \mathcal{C}_G(K)$ and all $x \in H$.

Lemma 44 *Let G be a chordal graph and $H \in \mathcal{C}_G(K)$. Then, $P_{i(H)}(H)$ separates H from $W = \mathcal{X}_{out}(V) \setminus P_{i(H)}(H)$ and is a minimal separator of G.*

Proof The set $P_{i(H)}(H)$ is a proper subset of all previously visited vertices (as H is not part of the *maximal* clique K Algorithm 4 starts with). Since $P_{i(H)}(H)$ contains all visited neighbors of H, it separates H from W. To see this, assume for sake of contradiction that there is a path from $v \in V_H$ to $w \in W$ without a vertex in $P_{i(H)}(H)$. Consider the shortest such path and let y be the first vertex with successor z preceding it in the vertex ordering produced by Algorithm 4 : $v - \cdots - x - y - z - \cdots - w \in W$. Then $\{x, z\} \in E_G$, as the ordering is a reverse of 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 $P_{i(H)}(H)$ is indeed a separator.

We prove that there is a vertex in W, which is a neighbor of all vertices in $P_{i(H)}(H)$. Consider the vertex in $P_{i(H)}(H)$, 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., $\mathcal{P}_{i(H)}(H) \setminus \{p\}$), meaning that p would have the same label. 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 H, which is a contradiction.

Hence, such vertex x has to exist. Moreover, x has to be connected to all vertices in $P_{i(H)}(H)$ 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 $P_{i(H)}(H)$. As both x and y are fully connected to $P_{i(H)}(H)$, it follows that this set is also a *minimal* $x - y$ separator. ■

Lemma 45 *Let G be a chordal graph for which the number of AMOs is computed with the function* `count` *in Algorithm 5. Let H be any chordal graph for which* `count` *is called in the recursion (for $H \neq G$). Then $H = F \setminus S$ for some S-flower F in G with $S \in \Delta(G)$.*

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

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

In the base case, $H \in \mathcal{C}_G(K)$. By Lemma 44, S_H is a minimal separator in G, which is by definition connected to all vertices in H. Hence, as H is connected, $H \subseteq F \setminus S_H$ 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 H. Then v can neither be a vertex in W nor the neighbor of a vertex in W, as by the definition of flowers this means that there is a path from W to H in $G[V \setminus S_H]$ – this would violate that H is separated from W by S_H (Lemma 44). Moreover, v is a neighbor of all vertices in S_H. Hence, we have $P_{i(v)}(v) = P_{i(H)}(H) = S_H$ 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 G'. Moreover, $G' = F' \setminus S_{G'}$ for some F'-flower of $S_{G'}$. Now, $P_{i(H)}(H)$ is by Lemma 44 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 $S_H = S_{G'} \cup P_{i(H)}(H)$ is a minimal x-y separator in G. Furthermore, S_H is fully connected to H and it can be easily seen that $H \subseteq F \setminus S_H$. To show equality, observe that every vertex v in $F \setminus S_H$ is in G' (if it is not separated from H by S_H, it is clearly not separated from H in $S_{G'}$). Thus, the same argument as in the base case applies and the statement follows. ■

Proof (of Proposition 27) By Lemma 45, 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 38 the flowers partition the bouquet and, by the 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 $2(|\Pi(G)| - 1) + 1 = 2|\Pi(G)| - 1$. ■

Acknowledgments

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

The authors would like to express their gratitude to Paula Arnold for fruitful discussions regarding uniform sampling of AMOs and her help in the implementations and the setup of the experiments. The authors also thank the anonymous reviewers and the action editor for their constructive comments and suggestions.

References

Ali AhmadiTeshnizi, Saber Salehkaleybar, and Negar Kiyavash. Lazyiter: A fast algorithm for counting Markov equivalent DAGs and designing experiments. In *Proceedings of the 37th International Conference on Machine Learning, ICML '20*, pages 125–133, 2020.

Steen A. Andersson, David Madigan, and Michael D Perlman. A characterization of Markov equivalence classes for acyclic digraphs. *The Annals of Statistics*, 25(2):505–541, 1997.

Anne Berry, Richard Krueger, and Genevieve Simonet. Maximal label search algorithms to compute perfect and minimal elimination orderings. *SIAM Journal on Discrete Mathematics*, 23(1):428–446, 2009.

Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer, 1993.

Graham R. Brightwell and Peter Winkler. Counting linear extensions is #P-complete. In *Proceedings of the 23th Annual ACM Symposium on Theory of Computing, STOC '91*, pages 175–181, 1991.

David Maxwell Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002a.

David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002b.

Derek G Corneil and Richard M Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008.

Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1):71–76, 1961.

Frederick Eberhardt. Almost optimal intervention sets for causal discovery. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, UAI'08*, pages 161–168. AUAI Press, 2008.

Frederick Eberhardt, Clark Glymour, and Richard Scheines. On the number of experiments sufficient and in the worst case necessary to identify all causal relations among N variables. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI'05*, pages 178–184. AUAI Press, 2005.

Robert Ganian, Thekla Hamm, and Topi Talvitie. An efficient algorithm for counting Markov equivalent DAGs. In *Proccedings of the 34th AAAI Conference on Artificial Intelligence, AAAI'20*, pages 10136–10143. AAAI Press, 2020.

Robert Ganian, Thekla Hamm, and Topi Talvitie. An efficient algorithm for counting markov equivalent dags. *Artificial Intelligence*, 304:103648, 2022.

AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Elias Bareinboim. Budgeted experiment design for causal structure learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML '18*, pages 1719–1728, 2018.

AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Kun Zhang. Counting and sampling from Markov equivalent DAGs using clique trees. In *Proccedings of the 33th AAAI Conference on Artificial Intelligence, AAAI'19*, pages 3664–3671. AAAI Press, 2019.

Steven B. Gillispie and Michael D. Perlman. The size distribution for Markov equivalence classes of acyclic digraph models. *Artificial Intelligence*, 141(1/2):137–155, 2002.

Kristjan H. Greenewald, Dmitriy Katz, Karthikeyan Shanmugam, Sara Magliacane, Murat Kocaoglu, Enric Boix Adserà, and Guy Bresler. Sample efficient active learning of causal trees. In *Proceedings of the 32nd Conference on Neural Information Processing Systems, NeurIPS'19*, pages 14279–14289, 2019.

Alain Hauser and Peter Bühlmann. Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research*, 13:2409–2464, 2012.

Alain Hauser and Peter Bühlmann. Two optimal strategies for active learning of causal models from interventional data. *International Journal of Approximate Reasoning*, 55 (4):926–939, 2014.

Yang-Bo He and Zhi Geng. Active learning of causal networks with intervention experiments and optimal designs. *Journal of Machine Learning Research*, 9(Nov):2523–2547, 2008.

Yangbo He and Bin Yu. Formulas for counting the sizes of Markov equivalence classes of directed acyclic graphs. *arXiv*, abs/1610.07921, 2016. URL `http://arxiv.org/abs/1610.07921`.

Yangbo He, Jinzhu Jia, and Bin Yu. Counting and exploring sizes of Markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research*, 16(79):2589–2609, 2015.

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

Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5 (11):558–562, 1962.

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

Marloes H Maathuis, Markus Kalisch, and Peter Bühlmann. Estimating high-dimensional intervention effects from observational data. *The Annals of Statistics*, 37(6A):3133–3164, 2009.

Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, UAI'95*, pages 403–410, 1995.

Christopher Meek. *Graphical Models: Selecting Causal and Statistical Models*. PhD thesis, Carnegie Mellon University, 1997.

OEIS Foundation Inc. The number of irreducible permutations. entry a003319 in the on-line encyclopedia of integer sequences, 2022. URL `https://oeis.org/A003319`.

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

Emilija Perković, Johannes Textor, Markus Kalisch, and Marloes H. Maathuis. 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, 2017.

Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

Edward R. Scheinerman. Random interval graphs. *Combinatorica*, 8(4):357–371, 1988.

Oylum Seker, Pinar Heggernes, Tínaz Ekim, and Z. Caner Taskin. Linear-time generation of random chordal graphs. In *Proccedings of the 10th International Conference on Algorithms and Complexity, CIAC 17*, volume 10236, pages 442–453, 2017.

Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros G. Dimakis, and Sriram Vishwanath. Learning causal graphs with small interventions. In *Processing of the 28th Conference on Neural Information Processing Systems, NeurIPS'15*, pages 3195–3203, 2015.

Ilya Shpitser and Judea Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence, AAAI'06*, volume 21, pages 1219–1226. AAAI Press, 2006.

Ilya Shpitser, Tyler VanderWeele, and James Robins. On the validity of covariate adjustment for estimating causal effects. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, UAI'10*, pages 527–536. AUAI Press, 2010.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search, Second Edition*. MIT Press, 2000. ISBN 978-0-262-19440-2.

Chandler Squires, Sara Magliacane, Kristjan Greenewald, Dmitriy Katz, Murat Kocaoglu, and Karthikeyan Shanmugam. Active structure learning of causal DAGs via directed clique trees. In *Proceedings of the 33rd Conference on Neural Information Pressing Systems, NeurIPS'20*, volume 33, pages 21500–21511, 2020.

Topi Talvitie and Mikko Koivisto. Counting and sampling Markov equivalent directed acyclic graphs. In *Proccedings of the 33th AAAI Conference on Artificial Intelligence, AAAI'19*, pages 7984–7991. AAAI Press, 2019.

Robert E Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.

Benito van der Zander and Maciej Liśkiewicz. Separators and adjustment sets in Markov equivalent DAGs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI'16*, pages 3315–3321. AAAI Press, 2016.

Benito van der Zander, Maciej Liśkiewicz, and Johannes Textor. Separators and adjustment sets in causal graphs: Complete criteria and an algorithmic framework. *Artificial Intelligence*, 270:1–40, 2019.

Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence, UAI'90*, pages 255–270, 1990.

Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence, UAI'92*, pages 323–330, 1992.

Michael D Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on software engineering*, 17(9):972–975, 1991.

Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974.

Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Extendability of causal graphical models: Algorithms and computational complexity. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence, UAI'21*. AUAI Press, 2021a.

Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Polynomial-time algorithms for counting and sampling Markov equivalent DAGs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI'21*, pages 12198–12206. AAAI Press, 2021b.