UNIVERSITÄT ZU LÜBECK

From the Institute of Theoretical Computer Science
of the University of Lübeck
Director: Prof. Dr. rer. nat. Till Tantau

Algorithms for Markov Equivalence

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences and Technical Engineering

Submitted by
Marcel Wienöbst
from Hameln

Lübeck, 2023

First referee: Prof. Maciej Liśkiewicz
Second referee: Prof. Mathias Drton
Date of oral examination: 12.06.2024
Approved for printing: 27.08.2024

Abstract

Learning the causal relationships between random variables from observational data is a fundamental problem that pervades all empirical sciences. A popular way to model these relationships is through directed acyclic graphs (DAGs) and a central task in this context is to learn such models from data. Achieving this objective is complicated by the fact that there are classes of DAGs that are indistinguishable based on observations alone, known as *Markov equivalence classes* (MECs).

This thesis tackles fundamental algorithmic problems on the subject of Markov equivalence, in particular regarding the connection of the members of the MEC to its representation as *completed partially directed acyclic graph* (CPDAG), which is the output of causal discovery algorithms such as PC or GES. The main result is the first *polynomial-time* algorithm, which computes the size of the MEC for a given CPDAG. In turn, this also enables uniform sampling of MEC members in polynomial-time and improves the time complexity of further applications in causal discovery, regarding learning from interventional data and estimating causal effects over MECs.

These results are based on the study of *consistent extensions*, which formally are acyclic orientations of a partially directed graph that preserve its *v-structures* and which constitute a basic graphical notion in causal models. We investigate these objects from an algorithmic point-of-view, beginning with the problem of deciding whether a graph has a consistent extension, for which we present an algorithm improving the state-of-the-art and show its asymptotic optimality conditional on a fundamental complexity-theoretical conjecture. Building on this, we derive the first linear-time delay algorithm for listing all consistent extensions of a graph, a special case of this problem being the task of listing all members of an MEC.

Finally, we extend our studies to more expressive causal models allowing for unobserved confounding and selection bias, which are called maximal ancestral graphs (MAGs). We provide a new criterion for Markov equivalence of MAGs, which leads to an efficient algorithm for testing this relation improving upon previous work.

Zusammenfassung

Das Lernen der kausalen Zusammenhänge in einem System von Zufallsvariablen aus Beobachtungsdaten stellt ein fundamentales Problem in den empirischen Wissenschaften dar. Eine klassische Methode ist es, diese Zusammenhänge durch gerichtete azyklische Graphen (DAGs) zu modellieren, wobei es eine zentrale Aufgabe ist solche Modelle von Daten zu lernen. Letzteres wird dadurch erschwert, dass es Klassen von DAGs gibt, die auf Basis von Beobachtungen im Allgemeinen nicht zu unterscheiden sind, bekannt als *Markov-Äquivalenzklassen* (MECs).

Diese Arbeit befasst sich mit grundlegenden algorithmischen Problemen im Kontext der Markov-Äquivalenz, insbesondere in Bezug auf die Verbindung der Elemente der MEC zu deren Repräsentation als *completed partially directed acyclic graph* (CPDAG), welche das Resultat von Lernalgorithmen wie PC oder GES darstellt. Das Hauptergebnis ist der erste Polynomialzeit-Algorithmus, der die Größe der MEC für einen gegebenen CPDAG berechnet. Dies ermöglicht wiederum das uniforme Ziehen von Elementen der MEC in polynomieller Zeit und verbessert die Zeitkomplexität weiterer Anwendungen in der Kausalitätsanalyse, insbesondere bezüglich des Lernens auf Basis von Interventionsdaten und der Schätzung kausaler Effekte.

Diese Ergebnisse basieren auf der Untersuchung von *consistent extensions*, welche formal ausgedrückt azyklische Orientierungen eines teilweise gerichteten Graphen sind, die seine *v-structures* erhalten, und ein grundlegendes graphtheoretisches Konzept in der Kausalität darstellen. Wir untersuchen diese Objekte in einem allgemeinen Kontext, beginnend mit dem Problem zu entscheiden, ob ein Graph eine *consistent extension* besitzt, für das wir einen Algorithmus vorstellen, der bisherige Ansätze verbessert und dessen asymptotische Optimalität wir auf Basis einer fundamentalen komplexitätstheoretischen Vermutung zeigen. Darauf aufbauend leiten wir einen Algorithmus mit linearem *Delay* ab, um alle consistent extensions eines Graphen aufzulisten. Ein Spezialfall dieses Problems ist die Aufgabe, alle Elemente einer MEC aufzulisten.

Schließlich weiten wir unsere Analyse auf ausdrucksstärkere kausale Modelle aus, die *unobserved confounding* und *selection bias* zulassen und als *maximal ancestral graphs* (MAGs) bezeichnet werden. Wir stellen ein neues Kriterium für die Markov-Äquivalenz von MAGs vor, welches einen effizienten Algorithmus zum Testen dieser Relation ermöglicht.

Contents

1 Introduction **3**
- 1.1 Main Research Questions and Results . . . 5
- 1.2 Organization . . . 10
- 1.3 Acknowledgements . . . 11

2 Preliminaries **13**
- 2.1 Graphs and Basic Notation . . . 13
- 2.2 Bayesian Networks and Markov Equivalence . . . 15
- 2.3 Consistent Extensions of CPDAGs . . . 17

3 Extendability of Causal Graphical Models **23**
- 3.1 Background . . . 25
- 3.2 Lower Bounds . . . 27
- 3.3 Computing a Consistent Extension of a PDAG . . . 30
- 3.4 A Graphical Characterization of Extendable MPDAGs . . . 35
- 3.5 Recognition of Causal Graph Classes . . . 38
- 3.6 Application to Maximal Orientations of PDAGs . . . 40
- 3.7 Conclusions . . . 42

4 Enumerating Markov Equivalent DAGs **43**
- 4.1 Background . . . 45
- 4.2 Enumerating the Members of an MEC . . . 48
- 4.3 Enumerating Consistent Extensions of PDAGs . . . 52
- 4.4 Enumerating Markov Equivalent DAGs With Small Changes . . . 55
- 4.5 Experimental Evaluation . . . 59
- 4.6 Conclusions . . . 60

5 Counting Markov Equivalent DAGs **61**
- 5.1 Background . . . 62
- 5.2 The Clique-Picking Algorithm . . . 64
- 5.3 Experimental Evaluation . . . 81
- 5.4 Conclusions . . . 82

6 Applications of Efficiently Counting Markov Equivalent DAGs **83**
- 6.1 Uniform Sampling of Markov Equivalent DAGs . . . 84
- 6.2 Interventional CPDAGs and Active Learning . . . 89

6.3 Estimating Causal Effects from CPDAGs and Observed Data 94
6.4 Counting Consistent Extensions of PDAGs 97
6.5 Conclusions 99

7 Markov Equivalence of MAGs **101**
7.1 Background 102
7.2 Previous Work 103
7.3 A New Constructive Criterion for Markov Equivalence of MAGs 105
7.4 Algorithm for Testing Markov Equivalence of MAGs 108
7.5 A Different Approach to Markov Equivalence Testing 110
7.6 Related Problems 111
7.7 Experimental Evaluation 111
7.8 Conclusions 112

8 Conclusions **115**

Bibliography **117**

List of Symbols and Notation **125**

1

Introduction

Graphical models are an important tool for probabilistic reasoning in high dimensions. One of the most fundamental approaches is to use directed acyclic graphs (DAGs) for representing probability distributions, which is the basis of *Bayesian networks* [Pearl, 1988, Lauritzen, 1996]. These models also allow for an intuitive *causal* interpretation, with directed edges $x \to y$ encoding that x is a direct cause of y, and are the central building block for graphical causality [Pearl, 2009, Peters et al., 2017].

A DAG entails a set of *conditional independencies*, present in any distribution it can represent. Exploiting these independencies plays a crucial role in graphical models in general and in Bayesian networks in particular, where it, for example, enables approaches to learning the (causal) structure [Spirtes et al., 2000]. The notion of *d-separation* lies at the core of such methods, which is a purely graphical characterization of the conditional independencies implied by a DAG. Multiple DAGs may have the same d-separation relations, meaning they represent the same set of conditional independencies, which is known as *Markov equivalence*. This is a central obstacle in the task of identifying the *causal structure*, i.e., the causal DAG underlying the data, as Markov equivalent DAGs cannot be distinguished from observational data, at least based on the conditional independencies therein. Hence, algorithms for learning DAGs, such as PC [Spirtes et al., 2000] and GES [Chickering, 2002b], usually output a class of Markov equivalent DAGs instead of a single one, which is encoded by a *completed partially directed acyclic graph* (CPDAG).

This representation is based on the seminal result by Verma and Pearl [1990] that two Markov equivalent DAGs have the same *skeleton*, that is the graphs are identical when ignoring edge directions, and the same *v-structures*, which are induced subgraphs $a \to b \leftarrow c$. Consequently, *all* DAGs in a *Markov equivalence class* (MEC) share the same adjacencies and v-structures, both of which are preserved in the CPDAG representation. Generally, the CPDAG contains all directed edges which are fixed in the MEC (these can go beyond edges in v-structures, see Chapter 2 for details), while having an undirected edge $a - b$ if both $a \to b$ and $a \leftarrow b$ appear in DAGs in the class. In this way, the CPDAG is an efficient encoding of the class of DAGs, while also representing causal information through its directed edges. An example of a CPDAG and its MEC is given in Figure 1.1.

The CPDAG representation for an MEC is unique and it is possible to reconstruct the DAGs in the MEC by orienting the undirected edges in the CPDAG without creating a cycle or a new v-structure. Graphs with these properties are known as *consistent extensions* [Dor and Tarsi, 1992]. The duality between an MEC and its CPDAG representation is a central topic in this thesis. We tackle fundamental graphical problems in this domain with the goal of providing efficient and practical algorithms for causal reasoning under Markov equivalence (see Figure 1.2 for an overview, a more detailed introduction

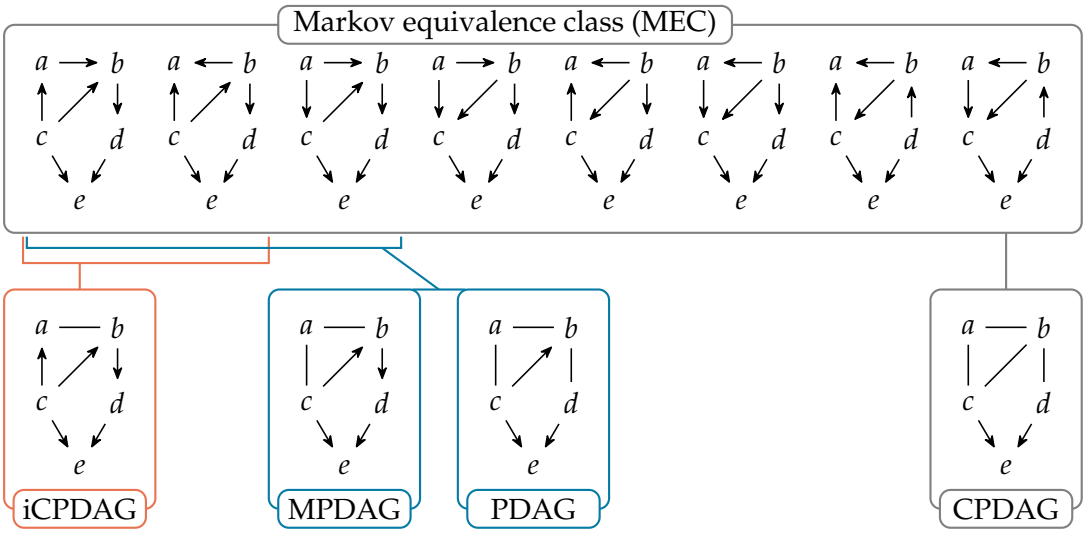


Figure 1.1: The Markov equivalence class represented by the CPDAG on the bottom right. The class contains all DAGs with the same skeleton and v-structures. Examples of an interventional CPDAG (iCPDAG) as well as an MPDAG and PDAG are displayed with the represented subclass of the MEC, i.e., the DAGs that are consistent extensions. The graph classes satisfy the inclusion relation CPDAG $\subset$ iCPDAG $\subset$ MPDAG $\subset$ PDAG, that is every CPDAG is also an iCPDAG and so on. iCPDAGs, MPDAGs and PDAGs represent subclasses of MECs, the latter two are equivalent in their expressive power (every subclass representable by a PDAG can be represented by an equivalent, but more informative MPDAG, e.g., the two example graphs in the figure have the same consistent extensions). Note that not every subclass of an MEC can be represented by a PDAG.

is given in the subsequent section). One example of such a task is to count the number of DAGs in an MEC given its CPDAG, for which we present the first polynomial-time algorithm.

CPDAGs have been generalized in multiple ways and we conduct our analysis under these different model classes. First, in case of additional background knowledge, for example from domain experts or due to time dependencies, it might be possible to determine further edge orientations. In this case as well, graphs with directed and undirected edges but without a directed cycle are used to represent the causal knowledge. Generally, we refer to such graphs as partially directed acyclic graphs (PDAGs). Another subclass of PDAGs, in addition to the class of CPDAGs discussed above, is known as *maximally oriented partially directed acyclic graphs* (MPDAGs for short) and is of interest as it represents maximally informative graphs under Markov equivalence in the presence of background knowledge (see Figure 1.1 for an illustration). The computational problems for PDAGs and MPDAGs are often harder compared to CPDAGs. For example, we show that, in contrast to CPDAGs, computing the size of the corresponding class of DAGs is #P-hard in these cases and thus intractable under standard complexity-theoretical assumptions.

Second, and closely related, edge orientations might be learned through means of intervention (experimentation) and in this case *interventional CPDAGs* (iCPDAGs in Figure 1.1), more commonly known as *interventional essential graphs*, constitute the corresponding model. These graphs have a specific structure, which often makes them more tractable compared to MPDAGs and we show that, e.g., the counting task is possible to solve in polynomial-time for these graphs, as is the case for CPDAGs.

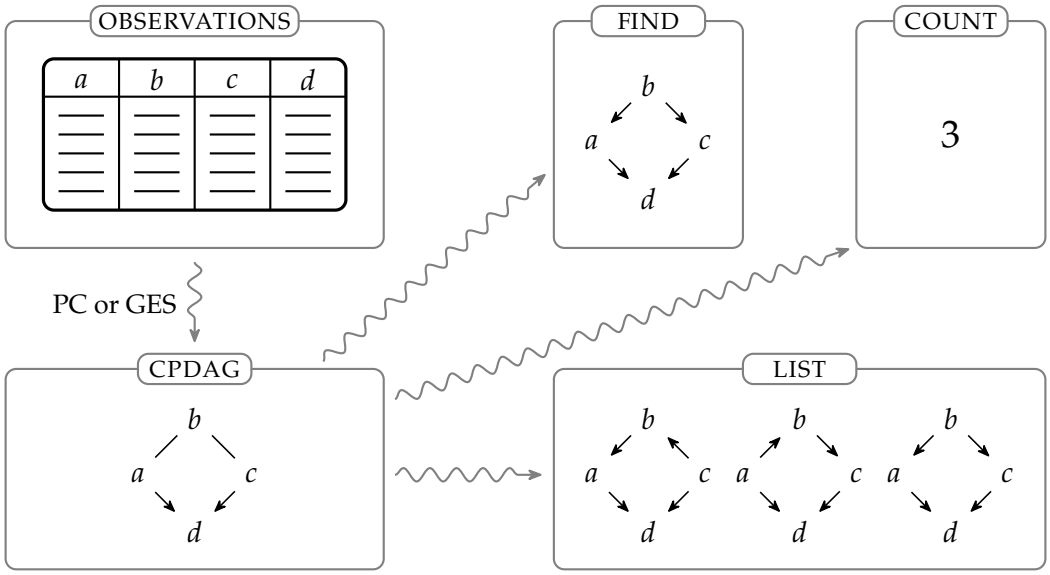


Figure 1.2: The CPDAG representation of an MEC is the starting point of our analysis as it is output by common causal discovery algorithms such as PC [Spirtes et al., 2000] or GES [Chickering, 2002b] in case of observational data. Basic reasoning tasks include finding a DAG from the corresponding MEC, listing all such DAGs and counting the number of them. The first task has a well-known solution for CPDAGs [Chickering, 2002a] discussed in Section 2.3 (for general PDAGs the matter is more intricate, see Chapter 3), the other two tasks are tackled in Chapter 4 and 5. For the example presented here, there are three DAGs in the MEC (shown under LIST) represented by the CPDAG on the bottom left. Note that the orientation $a \to b \leftarrow c$ introduces a new v-structures, thus implying that the corresponding DAG is *not* part of the illustrated MEC.

Third, DAGs (and CPDAGs) are unable to represent causal systems in case of unobserved variables, which is a large obstacle to the practical usage of causal-structure-learning algorithms. In this more general setting, *maximal ancestral graphs* (MAGs) are used, which can, similarly to DAGs, be connected to conditional independencies, again yielding the notion of Markov equivalence. The main drawback of using the MAG model is its additional complexity compared to DAGs. We aim to decrease this complexity gap in order to further facilitate practical usage of MAGs.

1.1 Main Research Questions and Results

The main topic of this thesis is to design efficient and practical algorithms as well as to provide a complexity-theoretical classification of the most fundamental tasks concerning Markov equivalence classes (or subclasses thereof) of DAGs. Formulated in general terms, given a PDAG, or more restricted subclasses such as (interventional) CPDAGs or MPDAGs, we study algorithmic questions regarding the corresponding classes of DAGs, which are, in graphical terminology, known as *consistent extensions* [Dor and Tarsi, 1992]. Figure 1.2 gives a high-level overview for the special case of CPDAGs.

Generally, we speak of *PDAGs*, whenever we do not want to further restrict or specify the model class. Nonetheless, we will still classify the complexity for every relevant graph class (e.g., CPDAGs, MPDAGs etc.) and note that certain problems (such as FIND-

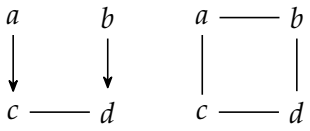


Figure 1.3: Two examples of PDAGs without a consistent extension. On the left, both orientations of $c - d$ yield a new v-structure. On the right, every orientation without a v-structure is cyclic.

ING described below) are intrinsically interesting and relevant for the class of PDAGs itself. For a given PDAG G, we study the three following basic algorithmic problems:

- FINDING (or constructing) a consistent extension of G or deciding that none exists.
- LISTING (or generating or enumerating) all consistent extensions of G.
- COUNTING the number of consistent extensions of G.

These three categories are fundamental axes in the study of combinatorial algorithms and complexity theory [Knuth, 2014]. As will (hopefully) become evident in this thesis, PDAGs and their consistent extensions make for a rich and interesting object from the complexity-theoretical and algorithmic point-of-view. FINDING is closest to the decision problems which underlie classes such as P and NP (however, all finding problems considered in this thesis are in P and our focus lies in studying and improving the asymptotic as well as the practical time complexity). The complexity of COUNTING is another well-studied branch in combinatorial algorithms and complexity theory. It has deep connections to uniform sampling, which we explore in Chapter 6. Finally, there is a large literature on LISTING algorithms, whose complexity is commonly measured according to their *delay*, that is the time between two successive outputs.

FINDING a consistent extension of a CPDAG, as well as an interventional CPDAG, is well-known to be solvable in linear-time in the size of the graph [Chickering, 2002a, Hauser and Bühlmann, 2012], which is asymptotically optimal. Our contributions for this task lie in the case of MPDAGs and general PDAGs. In the former, we show that it is, as for CPDAGs, possible to compute a consistent extension in linear-time. In the latter, the FINDING problem (or more precisely the decision problem whether a PDAG has a consistent extension) is classically part of the question whether give observational data admit a causal explanation. Verma and Pearl [1992] present an algorithm which first constructs a PDAG based on conditional independencies in the data. In the last step, it is checked whether the PDAG represents a non-empty set of DAGs, i.e., whether it has a consistent extension. Figure 1.3 shows examples where this is not the case. These concepts were further developed in constraint-based causal discovery, where PDAGs play a central role. The problem of finding a consistent extension of a PDAG or deciding that none exists, can be solved by a simple algorithm due to Dor and Tarsi [1992] in time[1] $O(n^4)$. However, the question remained whether faster algorithms can be constructed, for example achieving linear-time as for CPDAGs. Our main contribution lies in showing, on the one hand, that there is an $O(n^3)$ algorithm for computing a consistent extension of a general PDAG and, on the other hand, that under certain complexity-theoretical assumptions, it is *not* possible to solve this problem in linear-time.

Theorem 1.1. *Let G be a PDAG. There is an algorithm that decides whether G is extendable in expected time $O(dm)$. If G is extendable, a consistent extension can be computed within the same time bound. Here, d is the degeneracy of the skeleton of G.*

[1] Throughout this thesis, particularly when stating the time complexity of an algorithm, n and m denote the number of vertices and edges of the input graph.

Degeneracy formalizes the notion of sparsity and, for example, planar or constant-treewidth graphs have constant degeneracy. The lower bound below relies on the *Strong Triangle Conjecture*, formally introduced in Chapter 3, which postulates a lower bound on the complexity of finding three pairwise connected vertices in an undirected graph, a deeply studied algorithmic problem.

Theorem 1.2. *For any $\varepsilon > 0$, deciding whether a PDAG has a consistent extension in time $O(n^{3-\varepsilon})$ by a combinatorial or in time $O(n^{\omega-\varepsilon})$ by an algebraic algorithm would violate the Strong Triangle Conjecture.*

Here, $\omega < 2.3716$ denotes the *matrix multiplication exponent*, i.e., the smallest real number such that matrix multiplication can be performed in time $O(n^\omega)$. *Combinatorial* algorithms refer to ones, which do not rely on fast matrix multiplication. The distinction between combinatorial and algebraic algorithms is made because the latter are often not practical despite the better asymptotic bounds.

Using reductions to give conditional lower bounds makes it possible to connect graphical problems from causality to classical algorithmic problems. This connection provided a fruitful ground as it moreover yields that the recognition problem for MPDAG is not linear-time solvable under the same complexity-theoretical assumption.[2]

The lower bounds also extend to the problem of computing the *maximal orientation* of a PDAG, which is the corresponding MPDAG that represents the same class of DAGs. It has the property that it is its maximally informative representation, i.e., it has the largest number of directed edges of all graphs with the same consistent extensions as the given PDAG. Maximally orienting a PDAG is a fundamental primitive in causal discovery, occurring most prominently in the final step of the PC algorithm, but also in other algorithms especially in the context of active learning of DAGs [He and Geng, 2008, Hauser and Bühlmann, 2012]. Relying on and generalizing an idea given by Chickering [1995], we show that the algorithms for extending a PDAG can also be used to compute a *maximal orientation* of a PDAG. Hence, utilizing the $O(n^3)$ PDAG extension algorithm makes it possible to solve these problems in time $O(n^3)$ as well, which is significantly faster than classical approaches as later verified experimentally by Luttermann et al. [2023].

The LISTING problem certainly constitutes one of the most fundamental operations when dealing with Markov equivalence classes. Almost every basic reasoning task over MECs can be solved by considering all its members one-by-one, examples being counting the size of an MEC or sampling from it, and it is a building block of many causal inference and discovery software packages such as `pcalg` [Kalisch et al., 2012], `causaldag` [Squires, 2018], `TETRAD` [Ramsey et al., 2018] and `dagitty` [Textor et al., 2016]. The main drawback is the potential infeasibility of such an approach with an MEC having worst-case exponentially many members in the size of the CPDAG. This often necessitates specialized, more efficient algorithms not relying on enumeration, e.g., as we provide for the mentioned problems of counting and sampling in this work. Still, there might be cases where it is not possible or desirable to avoid a full enumeration of the MEC. For such cases, it is our aim to derive an enumeration algorithm, which is as fast as possible.

Despite its fundamentality, the problem has not been addressed in depth from the algorithmic point-of-view before. Enumeration approaches following from seminal results regarding the structure of Markov equivalence by Meek [1995] and Chickering [1995] are commonly used in practice. We approach the problem in a more direct way to significantly improve the state-of-the-art time complexity. For enumeration tasks, it is usually

[2] It can be shown that the task of *moralizing* a DAG, another fundamental task in probabilistic and causal reasoning, entails a similar hardness, proving common complexity analyses wrong [Wienöbst, 2023].

measured in terms of the delay, that is the time between two successive outputs. We give the first algorithm obtaining linear-time delay.

Theorem 1.3. *Let G be a CPDAG. There exists an algorithm which enumerates the members of the represented MEC with worst-case delay $O(n + m)$.*

In case every listed graph is output separately, the run-time is asymptotically optimal.[3] We also generalize this result to PDAGs and show that their consistent extensions can be listed with linear-time delay as well, after an $O(n^3)$ initialization step. Moreover, we give structural results showing that a Markov equivalence class can be enumerated in sequence such that two successive DAGs have distance at most three.

The task of COUNTING the number of DAGs in an MEC plays a central role in this thesis. This fundamental problem has a rich algorithmic history and it attracted many research groups because the basic question was unanswered whether it is possible to solve it in polynomial time in the size of the CPDAG [He et al., 2015, Talvitie and Koivisto, 2019, Ganian et al., 2020, 2022, AhmadiTeshnizi et al., 2020]. In the stated line of work, the algorithmic results were iteratively improved, however, in the worst-case, still amounted to exponential-time. In this thesis, we provide a positive answer to the above question by giving the *first* polynomial-time algorithm for this problem.

Table 1.1 contains an overview over all algorithmic results regarding consistent extensions and, in particular, shows these successive improvements culminating in the stated polynomial-time algorithm, which we name *Clique-Picking*:

Theorem 1.4. *Let G be a CPDAG. There exists an algorithm computing the size of the represented Markov equivalence class in time $O(n^4)$.*

In contrast, we show that counting the number of consistent extensions for general PDAGs is intractable in the sense that it is hard for the complexity class #P, which among other things means that a polynomial-time algorithm for this problem would imply that P equals NP. We still propose a generalization of the Clique-Picking approach to PDAGs by linking it to the task of counting the number of topological orderings of a DAG. While this algorithm exhibits worst-case exponential-time complexity, it is fast in many practical cases, in particular if the PDAG is sparse, respectively has small cliques. Independently, Sharma [2023] also build on Clique-Picking to tackle the general problem for PDAGs, studying its parameterized complexity [Downey and Fellows, 2012] and bounding the run-time with regard to a parameter again connected to the cliques of the graph.

From Theorem 1.4, it additionally follows that *uniformly sampling* a member from an MEC is also possible in polynomial-time, more precisely in linear-time $O(n + m)$ after initial preprocessing of time $O(n^4)$ (essentially amounting to performing the counting algorithm once). This, and further applications are discussed in Chapter 6, with the basic insight being that counting consistent extensions, while being hard for general PDAGs, is polynomial-time solvable for *interventional* CPDAGs. Hence, counting tasks in causal discovery from interventional data can be efficiently solved using the methods presented in this work as well. The relevance of these counting problems lies mainly in the field of designing experiments, also known as active learning [He and Geng, 2008, Hauser and Bühlmann, 2014], where it is a central goal to choose an intervention target which leads to the largest uncertainty reduction, respectively information gain (in worst-case or average-case), formalized through the number of DAGs explaining the given data.

[3] When modifying the graph in-place, faster delay is possible and sub-linear delay algorithms are an active area of research. It is an interesting open question whether such algorithms can be derived for the problem at hand.

Table 1.1: Results of this thesis put into the context of related work. This table includes the FINDING, LISTING and COUNTING problem for consistent extensions of PDAGs, MPDAGs and CPDAGs. As every MPDAG is a PDAG and every CPDAG is an MPDAG, every upper bound transfers and we hence omit repeating results, e.g. $O(n+m)$ delay enumeration without additional initialization follows for CPDAGs from the row for MPDAGs. Moreover, every result for CPDAGs also holds for interventional CPDAGs. Regarding notation, n indicates the number of vertices, m the number of edges, Δ the maximum degree, k the clique-knowledge (see Sharma [2023]), t the treewidth of the graph and $\mathrm{TO}(n)$ the time complexity of computing the number of topological orientations of an n-vertex DAG (this problem is #P-hard).

	Model	Complexity Result	Reference	
FINDING	PDAG	$O(n^4m)$ alg.	Verma and Pearl [1992]	CHAPTER 3
		$O(n^4)$ alg.	Dor and Tarsi [1992]	
		$O(n^3)$ alg.	Theorem 1.1	
		$\Omega(n^3)$ conditional lower bound	Theorem 1.2	
	MPDAG	$O(n+m)$ alg.	Theorem 3.27	
	CPDAG	$O(n+m)$ alg.	Chickering [2002a]	
LISTING	PDAG	$O(m \cdot n^3)$ delay alg.	Meek [1995]	CHAPTER 4
		$O(m^3)$ delay alg. (exp. space)	Chickering [1995]	
		$O(n+m)$ delay alg., $O(n^3)$ init.	Theorem 4.16	
	MPDAG	$O(n+m)$ delay alg., no init.	Theorem 1.3	
COUNTING	PDAG	$O(n^{\Delta+2})$ alg.	Ghassami et al. [2019]	CHAPTER 5
		Proof of #P-hardness	Theorem 6.9	
		$O(n^4 \cdot \mathrm{TO}(n))$ alg.	Theorem 6.12	
		$O(n^4 \cdot k!k^2)$ alg.	Sharma [2023]	
	CPDAG	$O(n!)$ alg.	Meek [1995], He et al. [2015]	
		$O(2^n \cdot n^4)$ and $O(t!2^t t^2 n)$ alg.	Talvitie and Koivisto [2019]	
		$O(2^n \cdot n^4)$ alg.	Ganian et al. [2020, 2022]	
		$O(2^{n+\Delta}(n\Delta+\Delta^3))$ alg.	AhmadiTeshnizi et al. [2020]	
		$O(n^4)$ alg.	Theorem 1.4	

Another observation is that interventional CPDAGs also play a role in estimating causal effects with regard to an MEC as was proposed by Maathuis et al. [2009]. In particular, every possible causal effect in an MEC corresponds to causal graphs represented by an interventional CPDAG, which enables efficiently counting the number of these DAGs and consequently weighting the causal effects in this way.

Finally, we extend our analysis of Markov equivalence to a more general class of graphs, namely *maximal ancestral graphs* (MAGs). Those graphs can express marginalized causal models, thus allowing for (implicit) latent variables as well as selection bias, which are both settings DAGs are unable to handle in the context of causal discovery. The additional expressivity of the MAG model, however, entails further complexity. In particular, Markov equivalence of MAGs is significantly less-well understood compared to DAGs. We initiate the study of decreasing this complexity gap between MAGs and DAGs. Our treatment of this topic is merely a starting point as we only deal with the task of deciding whether two MAGs are Markov equivalent, a problem which is almost

trivial for DAGs. We improve the time complexity of checking Markov equivalence of MAGs from $O(n^5)$ to $O(n^3)$ by providing a simplified graphical criterion:

Theorem 1.5 (Constructive-SRC)**.** *Let G_1 and G_2 be MAGs. They are Markov equivalent if, and only if,*

1. *G_1 and G_2 have the same adjacencies,*
2. *G_1 and G_2 have the same unshielded colliders, and*
3. *for all edges $b \leftrightarrow y \in G_1$ such that there is a discriminating path from a vertex x to y for b, it holds $b \rightarrow y \notin G_2$ and vice versa.*

1.2 Organization

This thesis contains further results beyond the central themes introduced above. Those will be put into context in the individual chapters, which contain individual introductions. Chapters 3 to 7 loosely correspond to the following first-author publications:

Chapter 3 [Wienöbst et al., 2021a] Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz: Extendability of Causal Graphical Models: Algorithms and Computational Complexity. In Proceedings of the 37th Conference in Uncertainty in Artificial Intelligence (UAI 2021). *Best Student Paper Award*.

Chapter 4 [Wienöbst et al., 2023] Marcel Wienöbst, Malte Luttermann, Max Bannach and Maciej Liśkiewicz: Efficient Enumeration of Markov Equivalent DAGs. In Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023).

Chapter 5 [Wienöbst et al., 2021b] 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 2021). *Distinguished Paper Award*.

Chapter 6 [Wienöbst et al., 2023] Marcel Wienöbst, Max Bannach and Maciej Liśkiewicz: Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs with Applications. In Journal of Machine Learning Research (JMLR) 24(213):1-45, 2023.[4]

Chapter 7 [Wienöbst et al., 2022] Marcel Wienöbst, Max Bannach and Maciej Liśkiewicz: A New Constructive Criterion for Markov Equivalence of MAGs. In Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI 2022). *Best Student Paper Award*.

The chapters are modified significantly compared to the original publications, mostly focusing on improving the presentation. This thesis is meant to be read as a monograph instead of five individual papers. Therefore, Chapter 2 contains general notation and well-known results relevant to multiple chapters. Chapter 8 concludes this thesis by summarizing the main contributions and discussing further open problems.

[4] This paper is an extended version of Wienöbst et al. [2021b], which is the basis of Chapter 5. Chapter 6 contains the additional parts in Wienöbst et al. [2023].

1.3 Acknowledgements

This thesis would not have been possible without the continual support from Maciej Liśkiewicz. Thank you, Maciej, for introducing me to causality research, for your guidance and advice, for letting me take my own path, while reminding of what's important, and for always sharing my enthusiasm and still keeping a critical eye at the same time.

I am grateful to all my co-authors and colleagues, for making the last four years so enjoyable. I learned a lot from all of you. In particular, I would like to thank Rüdiger Reischuk and Till Tantau for their trust and support, Max Bannach for all the enjoyable research we did together and Florian Chudigiewitsch for many inspiring discussions.

Finally, I would like to thank my family and friends for their love and support: Carmen, Carlo, Mark and Paula for always being there for me, and Clara for giving my life another dimension and everything else.

2

Preliminaries

In this chapter, we provide the notation used throughout this thesis and state fundamental results, which we rely on in other parts of this work. We start by defining basic graphical concepts. Then, we connect graphs to probabilities through Bayesian networks and introduce Markov equivalence in a formal way together with its graphical characterization. We conclude by recalling a linear-time algorithm for the problem of computing a member of a Markov equivalence class given its CPDAG [Chickering, 2002a].

All results in this chapter are known in the literature. We give references when appropriate, while taking the liberty to restate proofs in case they provide valuable insights.

2.1 Graphs and Basic Notation

We consider different types of graphs. Our standard model $G = (V, E, A)$ contains a set V of vertices, a set E of undirected edges $x - y$, a set A of directed edges $x \to y$ (also called arcs) and is referred to as *partially directed graph*. However, we often just write *graph* in this standard case. As special cases, we have *undirected graphs* ($A = \emptyset$) or *directed graphs* ($E = \emptyset$) and here we sometimes write $G = (V, E)$ instead of $G = (V, E, \emptyset)$ and $G = (V, A)$ instead of $G = (V, \emptyset, A)$. In Chapter 7, we additionally include bidirected edges into our model, denoted by $x \leftrightarrow y$ and given by a set B, yielding models $G = (V, E, A, B)$ and $G = (V, A, B)$. For all graphs, we demand that there is at most a single edge between any pair x, y of vertices and no self-loops, i.e., edges which connect a vertex to itself.

If there is an edge between vertices x and y (no matter which type), they are called *neighbors* or *adjacent*, denoted $x \sim y$. We write $x \sim_G y$ when explicitly referring to graph G in this and analogous settings. In case of the edge $x \to y$, vertex x is a *parent* of y and y is a *child* of x. Vertex x and y are *siblings* in case of edge $x \leftrightarrow y$, while we will simply speak of *undirected neighbors* in case of $x - y$. A path is a sequence of distinct vertices $v_1, v_2, \ldots, v_p$ such that v_i and v_{i+1} are adjacent for $i \in \{1, 2, \ldots, p-1\}$. A path is called *directed* (or *causal*) if all edges are directed $v_i \to v_{i+1}$. In case there is a directed path from x to y, then x is called an *ancestor* of y and y a *descendant* of x. Vertices are descendants and ancestors of themselves, but not parents/children. The sets of neighbors, undirected neighbors, parents, children, siblings, ancestors, and descendants of a vertex v are denoted by $Ne(v)$, $Un(v)$, $Pa(v)$, $Ch(v)$, $Si(v)$, $An(v)$, and $De(v)$, and they generalize to sets as exemplified for neighbors: $Ne(S) = \bigcup_{s \in S} Ne(s)$. We denote the *degree*, that is the number of neighbors, of vertex v by $\Delta(v)$ and more specifically the number of parents, children and undirected neighbors by $\delta^-(v)$, $\delta^+(v)$ and $\delta(v)$. The degree Δ of a graph is given by $\max_{v \in V} \Delta(v)$. Vertex v_i (for $1 < i < p$) on path $v_1, v_2, \ldots, v_p$ is called a *collider* if it is connected to its predecessor and successor as $v_{i-1} \to v_i \leftarrow v_{i+1}$, else it is called a non-collider. A cycle is a

sequence of vertices $v_1, v_2, \ldots, v_p$ such that (i) $v_1 = v_p$, (ii) $v_1, v_2, \ldots, v_p$ are distinct, (iii) $p > 2$ and (iv) v_i and v_{i+1} are adjacent for $i = 1, 2, \ldots p-1$. A cycle is *directed* if $v_i \rightarrow v_{i+1}$ for $i = 1, 2, \ldots p-1$. *Directed acyclic graphs* (DAGs) are directed graphs without a directed cycle.

The *skeleton* of graph $G = (V, E_G, A_G)$, denoted $\text{skel}(G)$, is the undirected graph $S = (V, E_S, \emptyset)$ with $E_S = \{\{u, v\} \mid u \sim_G v\}$. The *undirected subgraph* of graph G, denoted by $U(G)$, is defined as (V, E_G), i.e., it is the graph obtained by discarding the directed edges. Directed graph $D = (V, \emptyset, A_D)$ is an *orientation* of graph $G = (V, E_G, A_G)$ if (i) $\text{skel}(D) = \text{skel}(G)$ and (ii) $A_G \subseteq A_D$. We denote *sequences*, e.g. of vertices, as $(v_1, v_2, \ldots, v_n)$. The concatenation of sequences τ and π is given by $\tau + \pi$. Every sequence induces a *linear ordering* over its elements, namely the one where a is lesser than b if, and only if, it comes before b in the sequence. We often use the terms sequence and linear ordering interchangeably. We also consider *partial orders* $\preceq$ where only certain pairs of elements a and b are ordered ($a \preceq b$ or $b \preceq a$) such that the order is reflexive, antisymmetric and transitive. Given a linear ordering τ of the vertices of G, we denote the operation of orienting each edge $a - b$ in G according to τ, that is $a \rightarrow b$ if a comes before b in τ, as $G[\tau]$. Every DAG D has a linear ordering τ such that $\text{skel}(D)[\tau]$ equals D itself and it is called a topological ordering. The *induced subgraph* of G, for a set S of vertices, has vertex set S and contains all edges from G which have both endpoints in S. It is denoted by $G[S]$. A connected component of an undirected graph is a maximal induced subgraph such that all pairs of vertices are pairwise connected by a path. The set of *connected components* of G are denoted by $\mathcal{C}(G)$ and refers to maximal subgraphs pairwise connected by *undirected edges*. A set of vertices is a *clique* if they are pairwise adjacent.[1] It is a *maximal clique* if no proper superset is also a clique. If there exist vertices u and v such that vertex set S separates u and v (that is every path between u and v intersects S) and no proper subset of S separates u and v, then S is called a *minimal separator*. The set of maximal cliques of a graph G is denoted $\Pi(G)$ and the set of minimal separators $\Sigma(G)$. Note that there may exist minimal separators S and S' in G such that $S \subset S'$. A triple (a, b, c) of vertices in graph G is a *v-structure* if $a \rightarrow b \leftarrow c$, with $a \not\sim b$, in G. The set of v-structures of G is denoted by $\text{vs}(G)$. If $\text{vs}(G) = \emptyset$, then G is called *moral*.

Undirected graphs are *chordal* if every cycle of length ≥ 4 contains a chord, that is an edge between two vertices of the cycle, which is not part of the cycle. Chordal graphs can also be characterized as the graphs that have perfect elimination orderings (PEOs). A *perfect elimination ordering*[2] is a linear ordering τ of the vertices of graph G such that, for every vertex v, the neighbors of v coming before it in τ form a clique in G.

Chordal graphs can be represented by a clique-tree. A *rooted clique tree* of a connected chordal graph $G = (V_G, E_G)$ is a tree $T^R = (\Pi(G), E)$ where $R \in \Pi(G)$ is the root of the tree and, for any $v \in V_G$ and the set of cliques $\Pi^v(G) = \{K \in \Pi(G) \mid v \in K\}$, the induced subgraph $T^R[\Pi^v(G)]$ is connected. It is well-known that (i) every chordal graph has a rooted clique tree T^R 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 cliques $K, K' \in \Pi(G)$ in T^R with $K \cap K' = S$ [Blair and Peyton, 1993].

[1] Note that we define connected components as subgraphs and cliques as subsets of vertices. This simplifies the notation particularly in Chapter 5.

[2] PEOs are commonly defined such that the neighbors of v coming *after* it in the ordering should be a clique. The definition in the main text is more convenient for our purposes and conceptually equivalent.

2.2 Bayesian Networks and Markov Equivalence

We denote random variables by capital letters, such as X, and the values they take on by lowercase letters, such as x. We write $\mathbb{P}(x)$ as shorthand for $\mathbb{P}(X = x)$. Moreover, $(X \perp\!\!\!\perp Y \mid Z_1, \ldots, Z_k)_{\mathbb{P}}$ states that X and Y are *conditionally independent* given variables $Z_1, \ldots, Z_k$ and, conversely, $(X \not\perp\!\!\!\perp Y \mid Z_1, \ldots, Z_k)_{\mathbb{P}}$ that they are *conditionally dependent* with respect to probability distribution $\mathbb{P}$.

In *Bayesian networks*, DAGs are used to represent multivariate probability distributions. More precisely, a Bayesian network consists of a DAG $G = (V, A)$ a set of random variables $\{X_v \mid v \in V\}$ and conditional probability distributions $\mathbb{P}(x_v | x_{Pa(v)})$ with $X_{Pa(v)}$ denoting the set of random variables associated with the parents of v in G and $x_{Pa(v)}$ their values. It encodes the joint distribution

$$\mathbb{P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathbb{P}(x_i \mid x_{Pa(v)}).$$

A probability distribution $\mathbb{P}$, which factors in this way with regard to a DAG G, is called *Markovian* to G. Conditional independence lies at the core of Bayesian networks. Graphically, it is captured by d-separation:

Definition 2.1 (d-separation)**.** *Let $G = (V, A)$ be a DAG, $a, b \in V$ and $C \subseteq V \setminus \{a, b\}$. Then, a is said to be* d-separated *from b given C in G if every path π between a and b contains a vertex p such that*

1. *$p \in C$ and p is a non-collider on π or*
2. *$De(p) \cap C = \emptyset$ and p is a collider on π.*

In this case, we write $(a \perp\!\!\!\perp b \mid C)_G$, else we say a and b are d-connected*, which is denoted by $(a \not\perp\!\!\!\perp b \mid C)_G$.*

The justification for this definition is the following result.

Theorem 2.2 (Pearl [1988], Lauritzen and Wermuth [1989])**.** *Let $G = (V, A)$ be a DAG and $\mathbb{P}$ be a joint probability distribution over random variables $\{X_v \mid v \in V\}$, which is Markovian to G. Then, it holds for vertices a, b and a disjoint set of vertices C that*

$$(a \perp\!\!\!\perp b \mid C)_G \implies (X_a \perp\!\!\!\perp X_b \mid X_C)_{\mathbb{P}}.$$

Notably, the reverse direction does not hold generally, but there exist probability distributions for which it does (those are called *faithful*). The *faithfulness assumption* is central in recovering the structure of a Bayesian network as it asserts that the conditional independencies in the data are identical to the d-separation relations in the underlying DAG. These observational quantities, however, are not sufficient for uniquely identifying this DAG, which is the goal in *causal discovery*. E.g., the following DAGs entail the same d-separation relations and hence imply the same set of conditional independencies: $a \to b \to c$, $a \leftarrow b \leftarrow c$ and $a \leftarrow b \to c$. This is formalized and generalized through the notion of *Markov equivalence*.

Definition 2.3 (Markov equivalence)**.** *DAGs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are* Markov equivalent *if for all $a, b \in V$ and $C \subseteq V$ it holds that $(a \perp\!\!\!\perp b \mid C)_{G_1} \iff (a \perp\!\!\!\perp b \mid C)_{G_2}$.*

This definition is constructive in the sense that, algorithmically, the Markov equivalence of two DAGs can naively be decided by checking every possible independence statement using the d-separation criterion. However, there is a more concise and elegant graphical characterization:

Theorem 2.4 (Verma and Pearl [1990], Frydenberg [1990])**.** *Let G_1, G_2 be two DAGs. Then, G_1 and G_2 are Markov equivalent if, and only if,*

1. $\text{skel}(G_1) = \text{skel}(G_2)$,
2. $\text{vs}(G_1) = \text{vs}(G_2)$.

Clearly, Markov equivalence is an equivalence relation and thus partitions the space of DAGs into *Markov equivalence classes* (MECs):

Definition 2.5 (Markov equivalence class)**.** *Let D be a DAG. Then, the* Markov equivalence class *of D, denoted $[D]$ consists of all DAGs D' such that D' is Markov equivalent to D.*

Due to (1.), all DAGs in $[D]$ have the same skeleton, only the orientation of edges can differ. From (2.), it follows that, in case there is at least one v-structure, the DAGs in an MEC have some fixed directed edges. The graphical representation of a class of DAGs exploits these two facts. We introduce it here in a more general form:

Definition 2.6 (PDAG representation)**.** *The* PDAG representation *of a set of DAGs $\mathcal{C} = \{D_1, D_2, \dots\}$ with the same vertex set V is denoted* $\text{pdag}(\mathcal{C})$ *and given by graph $G = (V, E, A)$ with*

$$E = \{a - b \mid \text{there exist } D_i, D_j \in \mathcal{C} \text{ such that } a \to b \in E_i \text{ and } a \leftarrow b \in E_j\},$$
$$A = \{a \to b \mid \text{for all } D_i \in \mathcal{C} \text{ it holds that } a \to b \in E_i\}.$$

Definition 2.7 (CPDAG)**.** *Let $\mathcal{C}$ be a Markov equivalence class of DAGs. Then, the PDAG representation $G = \text{pdag}(\mathcal{C})$ is called* completed partially directed graph *(CPDAG) or* essential graph*. The MEC represented by CPDAG G is denoted by $[G]$.*

From Theorem 2.4, we directly obtain a precise link between CPDAG G and MEC $[G]$.

Corollary 2.8 (of Theorem 2.4)**.** *Let G be a CPDAG. Then, DAG D is in $[G]$ iff* $\text{skel}(G) = \text{skel}(D)$ *and* $\text{vs}(G) = \text{vs}(D)$.

Proof. Assume D is in $[G]$. Then, it has the same skeleton and v-structures as any DAG D' in $[G]$ by Theorem 2.4. By Definition 2.7, G consequently has the same skeleton and the same v-structures as D.

Conversely, assume that DAG D has the same skeleton and v-structures as G. Let D' be any DAG from $[G]$. Then, by the first part of the proof, D' has the same skeleton and v-structures as G and, by transitivity, of D. Hence, by Theorem 2.4, D is Markov equivalent to D' and thus also in $[G]$. □

This immediately yields the following obvious statement.

Lemma 2.9. *Let $G = (V, E_G, A_G)$ be a CPDAG and $D = (V, \emptyset, A_D)$ a DAG in $[G]$. Then, $A_G \subseteq A_D$.*

Proof. Assume, for the sake of contradiction, there exists an edge $a \to b$ in A_G, which is not in A_D. As $\text{skel}(G) = \text{skel}(D)$ by Corollary 2.8, it would follow that $a \leftarrow b$ is in A_D. But then, by Definition 2.7, $a - b$ in E_G and hence not $a \to b$ in A_G violating the assumption. □

To conclude this section let us state the following fundamental lemma about the local structure of CPDAGs.

Lemma 2.10 (Andersson et al. [1997], Meek [1995], Chickering [1995])**.** *Let G be a CPDAG. Then, it does* not *contain the induced subgraphs (i)* $a \longrightarrow b \longrightarrow c$ *, (ii)* $a \overset{\frown}{\longrightarrow} b \longrightarrow c$ *, (iii)* $a \overset{\frown}{\longrightarrow} b \longrightarrow c$ *and (iv)* $a \overset{\frown}{\longrightarrow} b \longrightarrow c$ *.*

Proof sketch. For (i), the statement follows because DAGs with $a \to b \to c$ and $a \to b \leftarrow c$ are in different MECs. Hence, the undirected edge $b - c$ cannot occur in a CPDAG by definition. For (iii), we have that $a \overset{\frown}{\longrightarrow} b \longrightarrow c$ cannot occur in a DAG, which by the same argument as above means $a - c$ cannot be in a CPDAG. For (iv), observe that it would imply the directed cycle to be in every DAG in $[G]$ by Lemma 2.9, which would violate acyclicity and hence the fact that $[G]$ is non-empty. The remaining case (ii) needs more technical effort, we therefore omit it here. For a proof, see Lemma 3 in [Chickering, 1995], Lemma 1 in [Meek, 1995], respectively the proofs in [Andersson et al., 1997]. □

There are other induced subgraphs CPDAGs cannot contain. However, as these are not necessary for the arguments in this chapter, we defer a discussion to Chapter 3.

2.3 Consistent Extensions of CPDAGs

As the conclusion of this chapter, we formally introduce one of the central combinatorial objects in this thesis: *consistent extensions*.[3] Definition 2.7 states how the CPDAG representation can be obtained given an MEC. The reverse operation, however, is more commonplace, that is to move from the CPDAG representation to the corresponding MEC of DAGs (or some properties of it), given that the CPDAG is directly learned from data through algorithms such as PC and GES [Spirtes et al., 2000, Chickering, 2002b], and not explicitly the full class of DAGs, which would be a more inconvenient representation. The basic tasks in this domain have been introduced in Chapter 1 (see e.g. Figure 1.2 for an overview). In this section we discuss the problem of finding a DAG in the MEC represented by a CPDAG G, which can be solved in linear time [Chickering, 2002a]. For this, we first derive well-known characterizations of the DAGs in the MEC represented by a given CPDAG based on Corollary 2.8 above. We will rely on these results throughout this thesis.

Definition 2.11 (Consistent Extension)**.** *Let G be a graph. Then, directed graph D is called a* consistent extension *of G if*

1. *D is an orientation of G,*
2. *D is acyclic and*
3. *D has the same v-structures as G, i.e.* $\text{vs}(G) = \text{vs}(D)$.

Building on Corollary 2.8, we have that:

Lemma 2.12. *Let $G = (V, E_G, A_G)$ be a CPDAG. Then, directed graph $D = (V, \emptyset, A_D)$ is in $[G]$ iff it is a consistent extension of G.*

[3] In this section, we only focus on consistent extensions of CPDAGs. The general case for PDAGs is investigated in the subsequent chapter. Notably, several of the useful properties of consistent extensions of CPDAGs do *not* hold for PDAGs in general, providing new challenges.

Proof. Assume D is a consistent extension of G. By Definition 2.11, it is (1.) an orientation of G and (2.) acyclic. Hence, it is a DAG with the same skeleton as G. Moreover, (3.) ensures that it has the same v-structures as G, which implies by Corollary 2.8 that D is in $[G]$.

Conversely, assume D is in $[G]$. Then, it is a DAG, thus satisfying condition (2.) in the definition of a consistent extension. By Corollary 2.8, $\text{vs}(G) = \text{vs}(D)$ implies (3.) and $\text{skel}(G) = \text{skel}(D)$ together with Lemma 2.9 implies (1.). □

While consistent extensions are defined for any PDAG G, we only consider the case that G is a CPDAG in this section. The general case, which entails additional challenges, is discussed in Chapter 3. In the remainder of this section, we discuss the algorithmic properties of the task of finding a consistent extension of a CPDAG.

Problem 2.13. FIND-CPDAG-EXT

Instance: *A CPDAG $G = (V, E, A)$.*
Result: *A consistent extension of G.*

This problem is well-studied and can be solved in linear-time in the size of the input graph [Chickering, 2002a, Hauser and Bühlmann, 2012]. The key towards this result lies in the following properties of the undirected subgraph $U(G)$ of a CPDAG G.

Lemma 2.14. *Let G be a CPDAG. If a and b are connected by an undirected path in G, then G does not contain $a \to b$.*

Proof. Assume there exists $a \to b$ with $a - p_1 - \cdots - p_k - b$ for $k \geq 1$ in G and assume this is a pair with a shortest such path. Then, $k = 1$ would imply the subgraph (ii) in Lemma 2.10, which does not occur in a CPDAG. For $k > 1$, we have $a \to b - p_k$. To avoid induced subgraph (i) of Lemma 2.10, a and p_k need to be adjacent. $a - p_k$ would immediately imply a shorter path and an arc $a \to p_k$ or $a \leftarrow p_k$ would imply a shorter path for pair a, p_k. □

Proposition 2.15 (Andersson et al. [1997], Meek [1995], He and Geng [2008])**.** *Let $G = (V, E, A_G)$ be a CPDAG and $U(G) = (V, E, \emptyset)$ its undirected subgraph. It holds that $C = (V, \emptyset, A_C)$ is a consistent extension of $U(G)$ iff $D = (V, \emptyset, A_G \cup A_C)$ is in $[G]$.*

Proof. For the first direction, consider $C = (V, A_C)$ and assume it is not a consistent extension of $U(G)$, even though $D = (V, \emptyset, A_G \cup A_C)$ is in $[G]$. Clearly, $\text{skel}(C) = U(G)$ holds (else D and G do not have the same skeleton violating Corollary 2.8). Hence, C contains a v-structure $a \to b \leftarrow c$ not in $U(G)$. Then, D also contains this v-structure, as there is no arc between a and c by Lemma 2.14. However, this v-structure is not in G implying that D is no consistent extension.

For the second direction, let C be a consistent extension of $U(G)$. It is immediate that (1.) D is an orientation of G. It remains to show that (2.) it is acyclic and (3.) that it has the same v-structures as G. We prove (3.) by contradiction. As clearly every v-structure in G is also in D, assume a *new* v-structure $a \to b \leftarrow c$ in D, which is not in G. Then, G contained either $a \to b - c$ or $a - b - c$ as induced subgraph, w.l.o.g., as D and G have the same skeleton by construction and $A_G \subseteq A_D = A_G \cup A_C$. By Lemma 2.10 (i), $a \to b - c$ cannot occur as induced subgraph in a CPDAG, whereas the induced subgraph $a - b - c$ in G (and hence in $U(G)$) would violate the assumption that C is a consistent extension of $U(G)$ as $\text{vs}(C) \neq \text{vs}(U(G))$.

For the acyclicity (2.), assume there is a directed cycle in D and consider the shortest one $c_1 \to c_2 \ldots c_p \to c_1$. Graph G itself does not contain a directed cycle, so there has

to be some $c_i - c_{i+1}$ in G with $c_{i-1} \to c_i$ (if this edge would not exist, the whole cycle would be undirected in G, contradicting the fact that C is acyclic by the definition of a consistent extension). We need to have an edge between c_{i-1} and c_{i+1} for $p \geq 4$, as by Lemma 2.10 (i) $c_{i-1} \to c_i - c_{i+1}$ does not occur as induced subgraph in a CPDAG. This edge is either oriented $c_{i-1} \to c_{i+1}$ or $c_{i-1} \leftarrow c_{i+1}$ in D. Both cases imply a shorter cycle and, thus, a contradiction. If the cycle has length three, i.e. $p = 3$, we simply distinguish four cases: If all edges in the cycle are directed in G or if one or two edges of the cycle are undirected, G is not a CPDAG by Lemma 2.10 (ii)-(iv). If all three edges are undirected, the acyclicity of C implies they do not form a cycle. □

Hence, for the task of constructing a consistent extension of a CPDAG G it suffices to compute a consistent extension of its undirected subgraph.[4] The directed edges in the CPDAG can be ignored in this process. As the consistent extensions of *undirected graphs* play such a central role, they are distinguished by their own terminology, which is justified by the following straightforward observation:

Lemma 2.16. *Let G be an undirected graph. Directed graph D is a consistent extension of G iff it is an* acyclic moral orientation *of G.*

Proof. Condition (1.) and (2.) of the definition of a consistent extension state it to be an acyclic orientation. Condition (3.) coincides with *morality* as the latter demands that $\mathrm{vs}(D) = \emptyset$ and, for G undirected, it holds that $\mathrm{vs}(G) = \emptyset$ in G and every consistent extension of it. □

We use *AMO* as shorthand for *acyclic moral orientation* and write $\mathrm{AMO}(G)$ for the set of AMOs of an undirected graph G. We will, throughout this work, prefer the term AMO over consistent extension for undirected graphs G, highlighting its *morality* (or v-structure freeness), even though there is no difference between those two notions in this case as certified by Lemma 2.16.

AMOs can be connected to PEOs as follows:

Lemma 2.17. *Let G be an undirected graph, τ a linear ordering of the vertices of G and $D = G[\tau]$. Then, D is an AMO of G iff τ is a PEO.*

Proof. Let τ be an PEO of G. We show that D is an AMO. It is obvious that it is an orientation of G and that it is acyclic. Assume there is a v-structure $a \to b \leftarrow c$ in D. Then, a and c come before b in τ but are non-adjacent, violating the assumption that τ is a PEO.

Let D be an AMO. As there is no v-structure in D, it holds for every vertex v that every pair of parents p_1 and p_2 of v is adjacent. The parents of v are the neighbors coming before it in τ and consequently they form a clique in G making τ an PEO. □

It immediately follows that:

Corollary 2.18. *An undirected graph G has an AMO iff it is chordal.*

Proof. Precisely chordal graphs have PEOs and by Lemma 2.17 a graph has a PEO if, and only if, it has an AMO. □

Figure 2.1 gives an intuition for the connection between AMOs and chordal graphs. Coming back to our original problem, we have that:

[4] We will use the other direction, that *every* DAG in $[G]$ can be obtained by finding a consistent extension of $U(G)$ in Chapter 4, where we tackle the task of efficiently listing all DAGs in $[G]$.

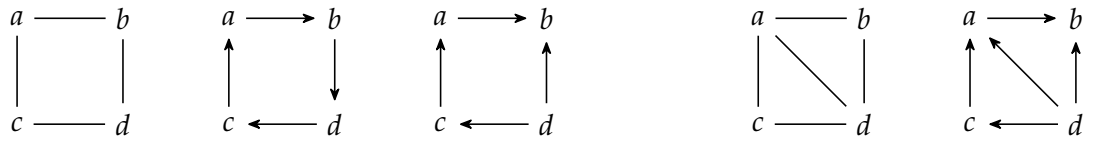


Figure 2.1: The graph on the left is a classic example of a *non*-chordal graph. It has no orientation which is both acyclic and moral. On the right, we show a modified graph, which contains a chord. This allows "breaking" the directed cycle without introducing a v-structure as $a \rightarrow b \leftarrow d$ is not an *induced* subgraph, there is the chord $d \rightarrow a$ as well.

Corollary 2.19. *Let G be a CPDAG and $U(G)$ the undirected subgraph of G. Then $U(G)$ is a chordal graph.*

Proof. For any CPDAG G, it holds that $[G]$ is non-empty by definition. It follows from Corollary 2.18 that a non-chordal subgraph has no AMO and thus no consistent extension (by Lemma 2.16). Then, by Proposition 2.15, the CPDAG would also have no consistent extension, which rules out the possibility of $U(G)$ being non-chordal. □

Hence, Problem 2.13 reduces to the following:

Problem 2.20. FIND-AMO

Instance: *A chordal graph $G = (V, E)$.*
Result: *An AMO of G.*

An algorithm for this task[5] can be used as a subroutine to solve Problem 2.13 following the arguments in this section. This approach is explicitly given in Algorithm 2.1 and Figure 2.2 illustrates these steps for an example graph.

Algorithm 2.1: Computing a consistent extension for CPDAG G (FIND-CPDAG-EXT).

input : CPDAG $G = (V, E, A_G)$.
output: $D \in [G]$.
1 $U(G) \coloneqq (V, E, \emptyset)$ `// U(G) is the (chordal) undirected subgraph of G`
2 $C \coloneqq$ `findamo(`$U(G)$`)` `// This function is implemented by Algorithm 2.2`
3 **return** $D \coloneqq (V, \emptyset, A_G \cup A_C)$ `// A_C refers to the set of arcs in C`

Thus, it remains to discuss how to tackle Problem 2.20. Lemma 2.17 and Corollary 2.18 already suggest that computing AMOs is intimately tied to the problem of chordal graph recognition. Indeed, every algorithm which is able to compute an AMO of an undirected graph G or decide that there exists none in time $O(T(n))$ can immediately be used to test chordality in time $O(T(n))$. Conversely, the most common algorithms for testing chordality of a graph G are implicitly connected to AMOs, as they proceed as follows [Rose et al., 1976, Tarjan and Yannakakis, 1984]:

[5]Note that we do not demand that U is connected here, even though it would be possible to include this restriction as each connected component of the undirected subgraph of G is chordal and can obviously be oriented independently to obtain a consistent extension of G. This constraint is not necessary (but would also not be harmful) for the methods described here. It will, however, be needed in Chapter 5, where we discuss it in more detail.

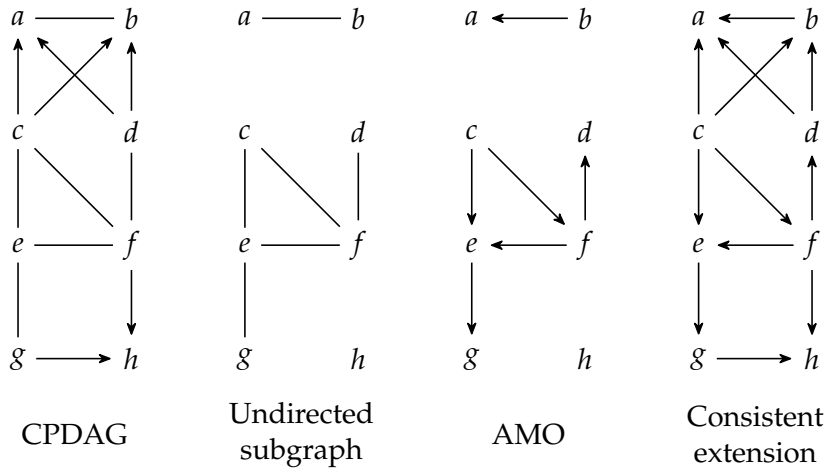


Figure 2.2: A CPDAG with its undirected subgraph and a possible AMO. Substituting it into the original CPDAG gives a consistent extension.

1. Compute an ordering of the vertices of G that is a PEO iff G is chordal.
2. Check whether the ordering is a PEO.

Thus, any algorithm for the first step can readily be used for our purpose of constructing an AMO of a chordal graph by orienting edges according to the PEO. Common algorithms for constructing the PEO are the Lexicographic Breadth First Search [Rose et al., 1976] and the Maximum Cardinality Search [Tarjan and Yannakakis, 1984], which both run in linear time in the size of the graph. We give the latter one in Algorithm 2.2, which solves FIND-AMO.[6]

The algorithm is, at its core, extremely simple. Choose the next vertex from the set of vertices with maximum number of visited neighbors. Implementing it in linear-time $O(n+m)$ is achieved by using appropriate data-structures (storing the number of visited neighbors for each vertex and, conversely, sets of vertices with the same number of visited neighbors) and updating these in $O(\Delta(x))$ after choosing vertex x. Let us make our discussion of this algorithm more precise:

- One *step* of the algorithm refers to one iteration of the for-loop in lines 4-11.
- $\tau_i = (v_1, \ldots, v_i)$ denotes the sequence of *visited vertices* after i steps of the algorithm. We write τ without subscript as shorthand for τ_n. A vertex not in τ_i is called an *unvisited vertex* (after i steps).
- $P_i(v) = Ne(v) \cap \tau_i$ denotes (in slight abuse of notation) the set of visited neighbors of vertex v at step i.
- $M_i = \{v \in V \setminus \tau_i \mid \text{for all } w \in V \setminus \tau_i \text{ it holds that } |P_i(v)| \geq |P_i(w)|\}$ is the set of unvisited vertices with maximum number of visited neighbors at step i.

[6]We will extensively build on the MCS algorithm later. For this task, and every other task in this thesis solved by MCS, one could also choose other graph traversal algorithms, whose visit order is a PEO for chordal graphs, under mild conditions. In Wienöbst et al. [2023] this is explored and the general framework of Maximum Label Search [Berry et al., 2009] discussed. In this thesis, we only consider MCS, mainly due to its simplicity.

Algorithm 2.2: Computing an AMO (FIND-AMO) of a chordal graph using Maximum Cardinality Search [Tarjan and Yannakakis, 1984].

input : Chordal graph $G = (V, E)$.
output: AMO D of G.

```
1  function mcs(G)
2      // τ_i is the sequence of visited vertices
3      τ_0 := ( )
4      for i ← 0 to n − 1 do                                   // n denotes |V|
5          // P_i(v) and M_i can be maintained more efficiently than stated
               in the pseudocode using appropriate data structures
6          foreach v ∈ V do P_i(v) := Ne_G(v) ∩ τ_i
7          M_i := {v ∈ V \ τ_i | for all w ∈ V \ τ_i it holds that |P_i(v)| ≥ |P_i(w)|}
8          // x is any vertex with maximum number of visited neighbors
9          x := any element of M_i
10         τ_{i+1} := τ_i + (x)                                // append x to τ_i
11     end
12     return τ_n
13 end
14 τ := mcs(G)
15 D := G[τ]          // Orient a − b in G as a → b if a comes before b in τ
16 return D
```

Proposition 2.21 (Tarjan and Yannakakis [1984])**.** *MCS (function* `mcs` *in Algorithm 2.2) visits the vertices in order of a PEO.*

Theorem 2.22. *For a graph with n vertices and m edges,* FIND-CPDAG-EXT *can be solved in time* $O(n + m)$.

Proof. Algorithm 2.1 computes a consistent extension of input CPDAG G: By Proposition 2.15, D is a consistent extension of G if C is a consistent extension, and thus, by Lemma 2.16, an AMO of $U(G)$. Corollary 2.19, asserts that graph $U(G)$ obtained by dropping the directed edges of G, is chordal. Hence, by Lemma 2.17 and because `mcs` in Algorithm 2.2 returns a PEO for chordal $U(G)$ (Proposition 2.21), `findamo` yields an AMO and the correctness follows.

It is well-known that MCS can be implemented in linear-time in the size of the graph. The remaining steps can trivially be implemented in $O(n + m)$. □

3

Extendability of Causal Graphical Models

One of the most fundamental tasks in the context of learning causal structures is to decide whether there may exist a causal explanation for given data. Verma and Pearl [1992] initiated systematic research in this direction by proposing an algorithm for deciding if a set of observed independencies over random variables may have an explanation expressed by a causal DAG. The algorithm first extracts as much information as possible from the independence statements and constructs a structure in form of a partially direct acyclic graph (PDAG). Next, the algorithm extends the PDAG to a consistent, i.e., Markov equivalent DAG, if the PDAG admits such a DAG extension. Finding consistent extensions turned out to be an important building block for causal discovery, also required by subsequent learning methods [Meek, 1995, Spirtes et al., 2000, Chickering, 2002a]. It is part of many software packages for causal analysis [Scutari, 2010, Kalisch et al., 2012]. In this chapter, we investigate this and related problems from an algorithmic and complexity-theoretical perspective.

The algorithm of Verma and Pearl [1992] finds a consistent DAG extension in time $O(n^4m)$, where n denotes the number of vertices and m the number of edges. Dor and Tarsi [1992] proposed a faster method of time complexity[1] $O(n^4)$; or $O(n^2\Delta^2)$ for PDAGs with maximum degree Δ. So far, it is the best-known algorithm for this problem and it is commonly used as a subroutine to solve more complex tasks. It is a long-standing open question whether the consistent DAG extension problem for PDAGs can be solved faster than in time $O(n^4)$, in particular, if it is solvable in time $O(n+m)$. Dor and Tarsi [1992] conjectured: "We believe that a linear-time chordality algorithm can be modified to a general linear-time algorithm for PDX[2]."

The main contributions of our work are two-fold.

- First, we propose a new algorithm for the extension problem for PDAGs which runs in time $O(n^3)$, or, more precisely, in time $O(\Delta m)$. Moreover, it solves the problem for d-degenerate[3] graphs in time $O(dm)$.
- Second, using fine-grained complexity analysis, we show that our algorithm is optimal under the *Strong Triangle Conjecture*.

Thus, under the above computational intractability assumption, the conjecture of Dor and Tarsi that there exists a linear-time algorithm to find a consistent DAG extension for

[1] Originally, $O(nm) \in O(n^3)$ was claimed incorrectly. Chickering [2002a] gave a refined analysis.
[2] PDX stands for the consistent extension problem for PDAGs.
[3] For a definition, see Section 3.3.

Table 3.1: Lower and upper bounds on the time complexity of the extension and recognition problem for PDAGs, CPDAGs, maximally oriented PDAGs, and Chain Graphs. Previously known bounds were proven by †: Dor and Tarsi [1992], *: Chickering [2002a] and ‡: Andersson et al. [1997]. Bounds in gray are trivial. Our results are given in a separate line. The novel lower bounds hold for combinatorial algorithms under the Strong Triangle Conjecture.

	Extendability		Recognition		
Model	Upper Bd.	Lower Bd.	Upper Bd.	Lower Bd.	Source
PDAG	$O(n^4)$		$O(n+m)$	$\Omega(n+m)$	†
	$O(n^3)$	$\Omega(n^{3-o(1)})$			Thm. 1.1, 1.2
CPDAG	$O(n+m)$	$\Omega(n+m)$		$\Omega(n+m)$	*
			$O(n+m)$		Prop.3.29
MPDAG	$O(n^4)$	$\Omega(n+m)$			†
	$O(n+m)$		$O(n^3)$	$\Omega(n^{3-o(1)})$	Thm. 3.27, 3.28, 1.2
CG	$O(n+m)$	$\Omega(n+m)$		$\Omega(n+m)$	‡
			$O(n+m)$		Prop. 3.29

PDAGs is not true. On the other hand, from our positive result it follows that, e.g., for forests, constant-treewidth, and planar graphs the problem can be solved in linear time.

We complete the investigation of the extension problem by proposing a linear-time algorithm for *Maximally Oriented PDAGs* (MPDAGs) [Meek, 1995, Perković et al., 2017], which form a subclass of PDAGs. The algorithm is based on a novel graphical characterization of extendable MPDAGs. Such models occur naturally when combining structures learned from observed data with background knowledge. They provide a useful framework for representing and analyzing sets of Markov equivalent DAGs.

With these results, we obtain the full and precise complexity-theoretic classification of the extension problem on various graphical causal models shown in Table 3.1. Note that linear-time algorithms were already known for CPDAGs [Andersson et al., 1997, Spirtes et al., 2000, Chickering, 2002a] and *Chain Graphs* (CGs) [Lauritzen and Wermuth, 1989, van der Zander and Liśkiewicz, 2016]. We discussed the former case in Section 2.3.

In order to leverage the linear-time algorithms for CPDAGs, MPDAGs and CGs, it is necessary to check whether a graph belongs to one of these classes. Hence, we expand our analysis to the recognition problem for these graphs. Interestingly, the lower-bound techniques can also be applied in this setting – yielding an $\Omega(n^{3-o(1)})$ bound for MPDAG recognition. We match this bound with an algorithm that recognizes MPDAGs in time $O(n^3)$. These results for the recognition problems are also stated in Table 3.1.

Finally, we combine our new algorithmic techniques to design an effective method for closing a PDAG under the orientation rules of Meek [1995] (for a definition, see Figure 3.1). This task of *maximally orienting* a PDAG, that is obtaining the MPDAG equivalent (with regard to its consistent extensions) to a given PDAG, is an important primitive used in algorithms for learning causal graphs.

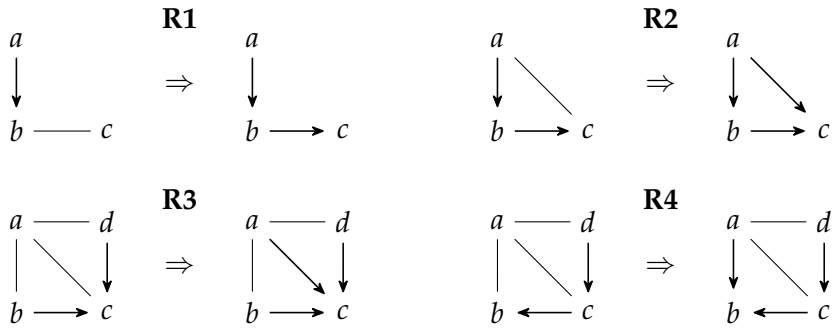


Figure 3.1: The four Meek rules that are used to characterize MPDAGs [Meek, 1995].

3.1 Background

Causal Graphical Models

Different classes of causal models are discussed in this chapter. In Chapter 2, only CPDAGs were formally introduced, which emerge naturally as the representation of MECs. Conversely, the notion of a consistent extension characterizes the DAGs represented by a CPDAG. Consistent extensions are formally defined for any graph $G = (V, E_G, A_G)$ (see Definition 2.11), not only CPDAGs. The only necessary assumption to give these consistent extensions a useful interpretation is that the skeleton and v-structures of the original graph are derived from the conditional independencies in observational (or interventional) data (hence, a consistent extension should not create more v-structures).

The most general graph class considered in this work are PDAGs, which are partially directed graphs which do not have a directed cycle (graphs with directed cycle do not have a consistent extension, hence they can be excluded). However, PDAGs still might not have consistent extensions and deciding whether this is the case is the main topic of this chapter. A standard interpretation of the PDAG model is that background knowledge can be incorporated, in the form of directed edges, which do not follow from v-structures. In a different setting, PDAGs occur in causal discovery algorithms such as PC, where first the skeleton and v-structures are discovered, which in itself does not yet yield a CPDAG (for this the rules in Figure 3.1 need to be applied).

CPDAGs have, by definition, the useful property that any undirected edge $a - b$ is oriented $a \to b$ in one consistent extension and $a \leftarrow b$ in another one. In that way the graph is maximally informative. General PDAGs do not have this property, however, *maximally oriented partially directed graphs* (MPDAGs) do. They are defined as follows:

Definition 3.1 (MPDAG)**.** *Let G be a graph. It is called* maximally oriented *or a* maximally oriented partially directed graph *(MPDAG) if it does not contain one of the induced subgraphs on the left side of "⇒" in Figure 3.1.*

We remark here that e.g. the graph on the left of Figure 2.1 satisfies this definition, while not satisfying the informal notion described in the text above: for every edge $a - b$ it is the case that neither $a \to b$ nor $a \leftarrow b$ are found in a consistent extension as there are none for this graph. Therefore, it often makes sense to consider *extendable* MPDAGs, which have a non-empty set of consistent extensions in addition to satisfying Definition 3.1.

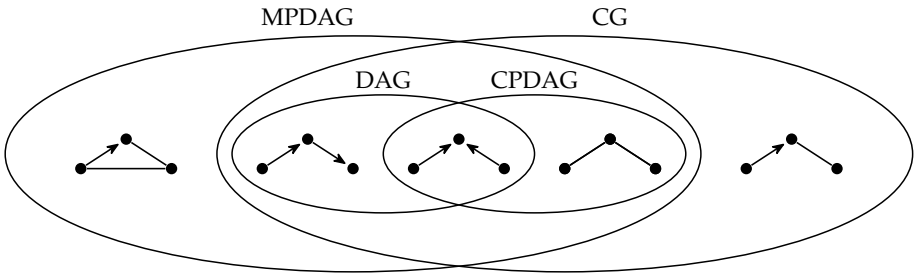


Figure 3.2: Relation between subclasses of PDAGs: MPDAGs, CGs, CPDAGs, and DAGs.

Chain graphs (CGs) are partially directed graphs that do not contain a semi-directed cycle, which is a cycle with at least one directed edge. It holds that every CPDAG is a chain graph Andersson et al. [1997]. More generally, how the graph classes relate to each other is shown in Figure 3.2.

Extendability From a Computational Perspective

In this chapter we focus on the computational aspects of extendability of causal graphical models. We not only ask the question whether a graph is extendable but to find a consistent extension if it is:

Problem 3.2. EXT
Instance: *A PDAG* $G = (V, E, A)$.
Result: *A consistent DAG extension of* G *if* G *is extendable; otherwise* $\perp$.

Restricting the instances to graphs of one of the previously introduced graph classes, we derive the problems MPDAG-EXT, CPDAG-EXT, and CG-EXT.

The class of extendable graphs can be seen as an intermediate class between chordal and acyclic graphs. The relation between these three classes can be stated as follows:

Proposition 3.3 (Dor and Tarsi [1992])**.** *Let* V *be a vertex set,* E *be a set of undirected edges, and* A *be a set of directed arcs. Then:*

$$(i) \qquad (V, E, \emptyset) \text{ is extendable} \iff (V, E) \text{ is chordal;}$$
$$(ii) \qquad (V, \emptyset, A) \text{ is extendable} \iff (V, A) \text{ is acyclic.}$$

Acyclicity can be tested in $O(n + m)$ as can chordality. Moreover, a consistent extension of a chordal graph can also be *constructed* in the same time complexity as discussed in the previous chapter. This implies that the above problem is linear-time solvable for these two boundary cases and it lead Dor and Tarsi [1992] to conjecture that the problem in general may be solvable in linear-time.

Another fundamental problem in this context is to recognize subclasses of PDAGs. We define CPDAG-REC and CG-REC analogously to:

Problem 3.4. MPDAG-REC
Instance: *A PDAG* $G = (V, E, A)$.
Question: *Is* G *an MPDAG?*

Finally, we consider the problem of computing the closure of a PDAG under the orientation rules of Meek:

Problem 3.5. MAXIMALLY-ORIENT
Instance: *A PDAG $G = (V, E, A)$.*
Result: *The graph obtained by exhaustively applying R1-R4 to G.*

If G is extendable, the resulting graph is unique and will, in general, be an MPDAG. In case all directed edges in G are implied by v-structure information (as for example in the oracle PC algorithm [Spirtes et al., 2000]), the resulting graph will also be a CPDAG [Meek, 1995]. On the other hand, if G is not extendable, different orders of applying the rules may lead to different graphs.

3.2 Lower Bounds

In this section, we derive lower bounds for MPDAG-REC, MAXIMALLY-ORIENT and the decision variant of EXT. In particular, the question of whether PDAGs can be extended in linear time has been debated in the past [Dor and Tarsi, 1992]. We show that in order to extend PDAGs for arbitrary E and A in linear time, however, a great algorithmic breakthrough would be necessary.

We prove these results through reductions from the problem TRIANGLE (does an undirected graph contain three pairwise connected vertices?). It is conjectured that, for any $\varepsilon > 0$, there is no algorithm that solves TRIANGLE in time $O(n^{\omega-\varepsilon})$ (were $\omega < 2.373$ is the matrix multiplication exponent) and no combinatorial algorithm running in time $O(n^{3-\varepsilon})$. Here, combinatorial means "not using algebraic techniques" and the distinction is made as algebraic algorithms are often not well-suited for practical applications[4].

Conjecture 3.6 (Strong Triangle Conjecture (STC))**.** *In the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $O(n^{\omega-o(1)})$ time in expectation to detect whether an n vertex graph contains a triangle. Moreover, any combinatorial algorithm requires time $O(n^{3-o(1)})$.*

Williams and Williams [2018] connected TRIANGLE to a whole class of problems, which thus all suffer from a cubic time barrier. A sub-cubic *combinatorial* algorithm for any of these problems would imply a sub-cubic algorithm for all of them. An important example in this class is the BOOLEAN-MATRIX-MULTIPLICATION problem (BMM).

We present $O(n^2)$ time reductions from the problem TRIANGLE to MPDAG-REC, MAXIMALLY-ORIENT and EXT. Hence, a linear-time algorithm for one of these problems (which runs in $O(n^2)$ as $m < n^2$), would imply an $O(n^2)$ algorithm for TRIANGLE. This would violate the Strong Triangle Conjecture no matter if such an algorithm is combinatorial or not, provided matrix multiplication cannot be performed in time $O(n^2)$. More generally, *any* sub-cubic combinatorial algorithm would give a sub-cubic combinatorial algorithm for TRIANGLE and in turn for problems such as BMM.

The reductions are based on the following idea: We partition the graph into three parts such that we have control over the structure of possible triangles; then we direct only a few edges such that detecting induced subgraphs $a \to b - c$ corresponds to finding triangles. Thus, we begin by the well-known statement that TRIANGLE is at least as hard as 3PART-TRIANGLE (does a 3-partite undirected graph contain a triangle?). An undirected graph is k-partite if there is a partition $V = V_1 \dot\cup V_2 \dot\cup \cdots \dot\cup V_k$ such that there is no edge $u - v \in E$ with $u, v \in V_i$ for some i.

[4]In the algorithmic literature, the term "combinatorial algorithm" is mainly used for distinguishing those approaches which are different from the algebraic approaches for fast matrix multiplication originating from the work of Strassen [1969]. For more details, see [Williams and Williams, 2018].

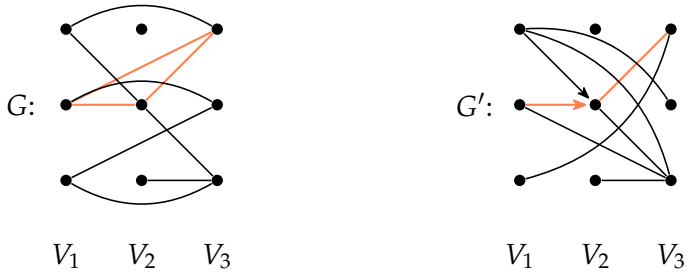


Figure 3.3: The basic step of the reductions: The edges between V_1 and V_2 are oriented towards V_2 and the edges between V_1 and V_3 are complemented. A triangle in G implies the subgraph $a \to b - c$ in G' and vice versa.

Lemma 3.7 (Folklore). *There is an $O(n^2)$ time reduction from* TRIANGLE *to* 3PART-TRIANGLE *that increases the number of vertices only by a constant factor.*

Proof. Let $G = (V, E)$ be an instance of TRIANGLE. Construct the graph $G' = (V_1 \cup V_2 \cup V_3, E')$ with $V_i = \{ v_i \mid v \in V \}$ and $E' = \{ u_i - v_j \mid u - v \in E \text{ with } i, j \in \{1, 2, 3\} \text{ and } i \neq j \}$, i.e., create three copies of the vertex set and insert the edges only between vertices in different copies.

We assume that G does not contain self-loops (as they are not part of any triangle anyway) and show that G contains a triangle if, and only if, G' contains one:

- $\Rightarrow$ Let $a, b, c \in V$ be a triangle in G. Then a_1, b_2, c_3 is a triangle in G'.
- $\Leftarrow$ Since G' is 3-partite, any triangle contains a vertex of each partition. Let a_1, b_2, c_3 be such a triangle; then the corresponding vertices $a, b, c \in V$ are mutually distinct due to the construction of G' and the lack of self-loops in G and, hence, form a triangle.

This reduction can be implemented in time $O(n + m)$ and increases the instance only by a constant factor. □

Building on this, we show a reduction from TRIANGLE to MPDAG-REC which proves the promised lower bound.

Lemma 3.8. *There is an $O(n^2)$ time reduction from* TRIANGLE *to the* MPDAG-REC *problem that increases the number of vertices only by a constant factor.*

Proof. First apply Lemma 3.7 to reduce TRIANGLE to 3PART-TRIANGLE and let $G = (V, E, \emptyset)$ be the resulting instance with partitions $V_1 \cup V_2 \cup V_3 = V$. Next, transform this instance as follows.

The basic idea is illustrated in Figure 3.3: From a 3-partite graph $G = (V_1 \cup V_2 \cup V_3, E, \emptyset)$ construct a graph $G' = (V_1 \cup V_2 \cup V_3, E', A')$ with

$$\begin{aligned} E' = &\{ u - v \mid u - v \in E \text{ with } u \in V_2 \text{ and } v \in V_3 \} \\ &\cup \{ u - v \mid u - v \notin E \text{ with } u \in V_1 \text{ and } v \in V_3 \}; \\ A' = &\{ u \to v \mid u - v \in E \text{ with } u \in V_1 \text{ and } v \in V_2 \}. \end{aligned}$$

Next, construct a graph $G_{\text{rec}} = (V, E' \cup \{ u - v \mid u \neq v \in V_1 \}, A')$, i.e., modify the construction shown in Figure 3.3 by additionally pairwise connecting V_1 with undirected edges. We show that G contains a triangle if, and only if, G_{rec} is not an MPDAG:

For the first direction, let a, b, c be a triangle in G and, w.l.o.g., assume $a \in V_1$, $b \in V_2$, $c \in V_3$. Then G_{rec} contains the induced subgraph $a \to b - c$ and is hence not an MPDAG, as R1 (Figure 3.1) would apply.

For the second direction, assume G_{rec} is not an MPDAG. Then at least one of the following statements is true: (1) R1 can be applied. (2) R2 can be applied. (3) R3 can be applied. (4) R4 can be applied. (5) G_{rec} contains a directed cycle.

Statement (5) does not hold as by construction there is no directed cycle. (4) does not hold as R4 needs a vertex c with $\delta^+(c) > 0$ and $\delta^-(c) > 0$, which does not exist in the construction. The same is true for vertex b in R2, hence (2) does not hold either. For R3, we need non-adjacent vertices b and d with $\delta^+(c) > 0$, hence, b and d have to be in V_1. But then, we have $b - d$. Hence, (3) does not hold. It follows that (1) has to apply and thus there is an induced subgraph $a \to b - c$ in G_{rec} (with $a \in V_1$, $b \in V_2$, $c \in V_3$ by construction) which corresponds to a triangle in G. □

From Lemma 3.8 we can deduce immediately that under the Strong Triangle Conjecture any combinatorial algorithm solving MPDAG-REC problem requires $\Omega(n^{3-o(1)})$ time. The same lower bound holds for MAXIMALLY-ORIENT. Otherwise, one could solve MPDAG-REC in sub-cubic time by applying the algorithm for MAXIMALLY-ORIENT to the given graph G and testing if the resulting graph coincides with G.

We now show that a similar reduction can be constructed for EXT as well:

Lemma 3.9. *There is an $O(n^2)$ time reduction from* TRIANGLE *to the decision variant of* EXT *that increases the number of vertices only by a constant factor.*

Proof. Apply Lemma 3.7 to reduce TRIANGLE to 3PART-TRIANGLE and let $G = (V, E, \emptyset)$ be the corresponding instance with $V_1 \cup V_2 \cup V_3 = V$. Construct graph $G' = (V \cup \{z\}, E', A')$ with

$$\begin{aligned} E' = &\{\, u - v \mid u - v \in E \text{ with } u \in V_3 \text{ and } v \in V_2 \,\} \\ &\cup \{\, u - v \mid u - v \notin E \text{ with } u \in V_3 \text{ and } v \in V_1 \,\} \\ &\cup \{\, u - v \mid u \neq v \in V_3 \,\}; \end{aligned}$$

$$\begin{aligned} A' = &\{\, u \to v \mid \ u - v \in E \text{ with } u \in V_1 \text{ and } v \in V_2 \,\} \\ &\cup \{\, z \to v \mid v \in V_3 \,\}. \end{aligned}$$

We show that G contains a triangle if, and only if, G' is not extendable:

$\Rightarrow$ Let $a, b, c \in V$ be a triangle in G with $a \in V_1$, $b \in V_2$, and $c \in V_3$. Then G' contains the induced subgraph $a \to b - c \leftarrow z$ and, hence, G' cannot be extended as either orientation of $b - c$ will create a new v-structure.

$\Leftarrow$ Assume there is no triangle in G. We construct an extension D' of G'. There are three types of undirected edges $x - y$, which we direct as follows:

1. $x \in V_1, y \in V_3$: Orient as $x \leftarrow y$.
2. $x \in V_2, y \in V_3$: Orient as $x \leftarrow y$.
3. $x \in V_3, y \in V_3$: Orient according to any linear ordering τ of V_3.

We show that this extension D' is acyclic and contains no new v-structure:

1. For the acyclicity, observe that we have $V_1 \to V_2 \leftarrow V_3 \leftarrow \{z\}$, with an arc $V_i \to V_j$ indicating that all edges between these vertex sets are oriented from $u \in V_i$ towards $v \in V_j$. The sets V_1 and V_2 are not adjacent to z. It is clear that any cycle would have to occur internally in a set V_i as a path $u \in V_i \to v \notin V_i \to \ldots \to w \in V_i$ cannot exist. But only the set V_3 contains internal edges, which are topologically ordered. Thus, D' is acyclic.
2. In the following case study, we analyze all situations in which $a \to b \leftarrow c$ can occur in D', as this is a necessary condition for a new v-structure. We show that either $a \sim_{G'} c$ or the v-structure already existed in G'.
   a) $a \in V_1, b \in V_2, c \in V_3$: A v-structure would be created if $a - c$ is not in G', but this implies a triangle in G.
   b) $a \in V_1, b \in V_2, c \in V_1$: These v-structures are present in G'.
   c) $a \in V_3, b \in V_1, c \in V_3$: We have $a \sim_{G'} c$ as V_3 is fully connected.
   d) $a \in V_3, b \in V_2, c \in V_3$: Same as (c).
   e) $a \in V_3, b \in V_3, c \in V_3$: Same as (c).
   f) $a \in V_3, b \in V_3, c = z$: V_3 and z are fully connected, hence $a \sim_{G'} z$.
   g) $a = z, b \in V_3, c \in V_3$: Symmetrical to (f). □

Our main result follows immediately from Lemma 3.8 and 3.9.

Theorem 1.2. *For any $\varepsilon > 0$, every algorithm that solves* MPDAG-REC, MAXIMALLY-ORIENT, *or* EXT *in time $O(n^{3-\varepsilon})$ by a combinatorial or in time $O(n^{\omega-\varepsilon})$ by an algebraic algorithm would violate the Strong Triangle Conjecture.*

3.3 Computing a Consistent Extension of a PDAG

We introduce a combinatorial algorithm for EXT that matches the lower-bound from the previous section. Furthermore, we show that the algorithm performs provably better on many important graph classes, e.g., we can compute extensions of graphs with bounded treewidth and of planar graphs in linear time.

A *graph property* is a family of graphs that is closed under isomorphism, e.g., being chordal is a graph property. We say that it is *hereditary* if it is also closed under taking induced subgraphs. For instance, chordality and acyclicity are hereditary graph properties. The same holds for extendability:

Lemma 3.10. *Extendability is a hereditary graph property. That is, if graph G is extendable, the same holds for any induced subgraph of G.*

Proof. Consider consistent extension D of G and S a subset of its vertices. Then, $D[S]$ is a consistent extension of $G[S]$: it is acyclic because that is a hereditary graph property and an orientation of $G[S]$ with the same v-structures by definition. □

Definition 3.11 (Elimination order)**.** *Let p be a property of a vertex. A p-*elimination-order *of a graph $G = (V, E, A)$ is an order π such that every vertex v has property p in the induced subgraph $G[V \setminus \{ w \mid \pi(w) < \pi(v) \}]$.*

Elimination orders characterize hereditary graph classes. For instance, a vertex v in an undirected graph is called *simplicial* if its neighbors $N(v)$ form a clique; a vertex w in a directed graph is a *sink* if it has no outgoing arc. It is well-known that a (directed)

graph is chordal iff it has a simplicial-elimination-order; and is acyclic iff it has a sink-elimination-order [Fulkerson and Gross, 1965]. Dor and Tarsi [1992] observed that these properties can be combined to obtain a characterization of extendable graphs:

Definition 3.12 (Potential-sink)**.** *A* potential-sink *in a graph $G = (V, E, A)$ is a vertex v such that $X = \{ w \mid v - w \in E \}$ is a clique, $\{ w \mid v \to w \in A \}$ is empty, and $x \sim_G y$ for all $x \in X$ and $y \in Y = \{ w \mid w \to v \in A \}$.*

Proposition 3.13 (Dor and Tarsi, 1992)**.** *Every partially directed graph that is extendable contains a potential-sink.*

Corollary 3.14. *A partially directed graph is extendable iff it has a potential-sink-elimination-order.*

Proof. The first direction follows from Proposition 3.13 as extendable graphs are hereditary. On the other hand, successively removing a potential-sink and orienting all its incident edges towards it yields an extension of the graph. □

Based on this characterization, Dor and Tarsi [1992] provided an $O(\Delta^2 m)$ algorithm for recognizing extendable graphs. We improve this result by presenting an algorithm that checks whether a graph is extendable in time $O(dm)$, where d is the *degeneracy* of the skeleton.

Definition 3.15 (Degeneracy)**.** *A graph $G = (V, E)$ is d-*degenerate *if there is a linear ordering of the vertices $\prec$ (called* degeneracy ordering*) such that $|\{ w \mid w \prec v \text{ and } w \sim_G v \}| \leq d$ for every $v \in V$. The smallest value d for which G is d-degenerate is the* degeneracy *of G.*

Observe that in any d-degenerate graph with n vertices, m edges, and maximum degree Δ we have $d \leq \Delta \leq n$ and $m \leq dn$. We dedicate this section to prove the following:

Theorem 1.1. *Let G be a PDAG. There is an algorithm that decides whether G is extendable in expected time $O(dm)$. If G is extendable, a consistent extension can be computed within the same time bound. Here, d is the degeneracy of the skeleton of G.*

The "expected time" in this statement comes from the fact that constant time adjacency tests in sparse graphs are necessary to achieve this run-time. Storing the neighbors of each vertex in a hash table, as we do in the methods presented here, yields expected time $O(1)$ for these operations, but not worst-case. Wienöbst et al. [2021a] show that the statement above holds for *worst-case* time $O(dm)$ using standard graph representation tricks, which however complicate the presentation and do not lend themselves well for a practical implementation, hence, we do not repeat those arguments here.

We assume that the undirected neighbors of vertex v as well as its children and parents are stored in three separate hash tables. At the basis of our algorithm is an efficient data structure for testing and updating whether a vertex is a potential sink. For each vertex v, we store in $\delta(v)$ the number of undirected neighbors, in $\delta^+(v)$ the number of children and in $\delta^-(v)$ the number of parents of v. Moreover, we manage the following information for each vertex v:

$$\alpha(v) = |\{ x - y \mid x - v \in E \wedge v - y \in E \wedge x \sim_G y \}|,$$
$$\beta(v) = |\{ x - y \mid x - v \in E \wedge y \to v \in A \wedge x \sim_G y \}|.$$

These values allows us to check potential-sinkness of a vertex s with a constant amount of arithmetic operations.

Table 3.2: Operations on graph $G = (V, E, A)$ and their (expected) run-time. For graph manipulation operations, this includes the update of the graph representation and data structure for maintaining potential sinks described in the text.

Method	(Expected) Time	Effect
`init`(n)	$O(n)$	Initialize $G = (\{v_1, \ldots, v_n\}, \emptyset, \emptyset)$.
`is-adjacent`(u, v)	$O(1)$	Return true if $u \sim_G v$.
`iterate-undir`(u)	$O(\delta(u))$	List all undirected neighbors of u.
`iterate-ch`(u)	$O(\delta^+(u))$	List all children of u.
`iterate-pa`(u)	$O(\delta^-(u))$	List all parents of u.
`insert-edge`(u, v)	$O(\Delta(u))$	Insert edge $u - v$ to E.
`insert-arc`(u, v)	$O(\Delta(u))$	Insert arc $u \to v$ to A.
`remove-edge`(u, v)	$O(\Delta(u))$	Remove edge $u - v$ from E.
`remove-arc`(u, v)	$O(\Delta(u))$	Remove arc $u \to v$ from A.
`is-ps`(s)	$O(1)$	Test whether s is a potential-sink.
`list-ps`$()$	$O(n)$	Return list of all potential-sinks in G.
`pop-ps`(s)	$O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$	Remove s from the graph. Return a list of neighbors of s, which became potential-sinks due to this operation.

Lemma 3.16. *A vertex $s \in V$ is a potential-sink iff:*

1. $\alpha(s) = \binom{\delta(v)}{2}$,
2. $\beta(s) = \delta(s) \cdot \delta^-(s)$ *and*
3. $\delta^+(s) = 0$.

Proof. Recall from Definition 3.12 that s is a potential-sink iff (i) $Un(s)$ is a clique, (ii) $x \sim_G y$ for all $x \in Un(s)$ and $y \in Pa(s)$, and (iii) $Ch(s)$ is empty. Obviously, item (iii) corresponds to the third item in the claim. For (i), we need $Un(s)$ to be fully connected, that is $G[Un(s)]$ to contain $\alpha[s] = \binom{\delta(s)}{2}$ edges. Finally, for (ii) we require that $Un(s)$ and $Pa(s)$ form a complete bipartite graph, i.e., we need $\delta(s) \cdot \delta^-(s)$ edges between them which coincides with $\beta(s) = \delta(s) \cdot \delta^-(s)$. □

We now discuss how this data structure can be maintained efficiently. Our goal is to achieve the run-times presented in Table 3.2. A few are obvious: Clearly, all the values for α and β are zero for the empty graph on n vertices, which allows for a trivial $O(n)$ initialization step (`init`). Lemma 3.16 implies that `is-ps`(s) and `list-ps`$()$ can be implemented in $O(1)$, respectively $O(n)$. Moreover, the run-time of `is-adjacent` and the `iterate` methods directly follow from the graph representation. We focus on how the `insert` and `remove` operations can be implemented in the stated time, as well as the `pop-ps` method.

Lemma 3.17. *Given a graph G with $Un(v)$, $Ch(v)$, $Pa(v)$ represented by hash tables and data structure $D = (\delta, \delta^+, \delta^-, \alpha, \beta)$, it holds that inserting or deleting edge $u - v$ or arc $u \to v$ in G and updating D accordingly is possible in expected time $O(\Delta(u))$.*

Proof. Adding edge $u - v$ or arc $u \to v$ to G is possible in expected time $O(1)$. It remains to show how D can be updated. We iterate over the neighbors of u (that is, all vertices x with $u \sim_G x$) and check whether $x \sim_G v$ holds as well (this is possible in expected time $O(1)$ due to hashing), i.e., we iterate over the common neighbors of u and v. If u, v, and x indeed constitute a triangle, all three vertices have a new edge in their neighborhood: x has the new edge $u \sim_G v$, u the new edge $x \sim_G v$, and v has $x \sim_G u$. Hence, we may *increase* the α- and β-values of all three vertices (of course, only if the edge directions match the definition of α and β). These updates require $O\big(\delta_1^+(u) + \delta_2^+(u) + \delta_2^-(u)\big)$ operations. Removing edges or arcs works analogously, we just have to *decrease* the corresponding α- and β-values. □

We also have to implement `pop-ps`(s), i.e., we wish to remove s and its incident edges from the graph and output all new potential sinks. This removal is "virtual", i.e., the universe size of our data structure remains n and we just mark s as "deleted" in the algorithm we discuss below. However, we *actually* delete all edges and arcs incident to s. The crucial part is to achieve this in time $\delta(s)^2 + \delta(s) \cdot \delta^-(s)$. Notably, we cannot directly use our `remove-edge` method, as this would use $O(\delta(s) + \delta^-(s))$ time *for every* incident edge.

Lemma 3.18. *Let G be a graph with $Un(v)$, $Pa(v)$, $Ch(v)$ represented by hash tables and data structure $D = (\delta, \delta^+, \delta^-, \alpha, \beta)$. The method `pop-ps`$(s)$ can be implemented in time $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$.*

Proof. We observe that updating α and β is significantly simpler if we remove an arc $v \to s$ instead of removing an arbitrary edge. In fact, we can ask which vertex x may have $v \to s$ in its neighborhood counted for its $\alpha[x]$- or $\beta[x]$-value. Clearly, we must have $s \sim_G x$ and, since $\delta^+(s) = 0$, we have either $s - x \in E$ or $x \to s \in A$. In the latter case, the removal of arc $v \to s$ affects neither $\alpha(x)$ nor $\beta(x)$. In contrast, the removal of edge $x - s \in E$ does. Therefore, to remove an arc $v \to s$, we iterate over the undirected neighbors of s (in $\delta(s)$) and for each such x we look-up (in constant time) the type of the edge (or arc) between x and v. Note that $x \sim_G v$ as s is a potential-sink. Depending on the type of this edge, we update the α- or β-value of each x and afterwards delete the arc $v \to s$ from the graph (in expected constant time). Using this trick, we can remove all arcs $v \to s$ in time $O(\delta(s) \cdot \delta^-(s))$ from the graph.

To achieve overall $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$, we remove the arcs $v \to s$ as described above *before* we remove the other edges incident to s. Once the arcs are gone, s has only undirected neighbors and, thus, we can remove the remaining edges in time $O(\delta(s)^2)$ using `remove-edge`. Finally, we iterate over the old neighbors of s and check whether they did become a potential-sink and, if so, output these vertices. □

The proof of Theorem 1.1 is based on the following slightly weaker statement, which only achieves time $O(\Delta m)$. It follows directly from the data structure described above.

Proposition 3.19. *There is an algorithm that decides whether a PDAG G is extendable in expected time $O(\Delta m)$.*

Proof. The backbone of the algorithm is the data structure described above with the runtimes displayed in Table 3.2. Given this tool, the algorithm becomes rather simple: Initialize the data structure by inserting all edges (in time $O(\Delta m)$); then initialize a stack of all potential-sinks by testing for every vertex whether it is a potential-sink (overall $O(n)$); finally, as long as there is potential-sink s on the stack, apply `pop-ps`(s), remove s from the stack, and push the newly generated potential-sinks to the stack. The overall run

Table 3.3: Running time of the algorithm from Theorem 1.1 on graphs whose skeleton is in the mentioned graph class.

Graph Class	Time	Note
d-degenerate	$O(dm)$	Shown in Theorem 1.1
general graphs	$O(nm)$	Follows as $d \leq n$
forests	$O(m)$	constant degeneracy
series-parallel	$O(m)$	
planar	$O(m)$	
treewidth-t	$O(tm)$	Follows as $d \leq t$

time is obtained as follows: `pop-ps`(s) runs in time $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$ and removes $\delta^-(s) + \delta(s)$ edges from the graph. By rewriting $\delta(s)^2 + \delta(s) \cdot \delta^-(s)$ as $\delta(s) \cdot \big(\delta(s) + \delta^-(s)\big)$ we see that we pay $O(\delta(s)) \leq O(\Delta)$ per edge. Since the algorithm terminates after the removal of all edges, an overall run time of $O(\Delta m)$ follows. □

In order to improve the run time of the algorithm from $O(\Delta m)$ to $O(dm)$, we need two things: First, a faster way of initializing the data structure (i.e., we cannot simply insert all m edges and pay $O(\Delta)$ per item), and we require a finer analysis of the `pop-ps` method. The main idea is to use the following facts about d-degenerate graphs: they always contain at least one vertex of degree at most d, and they do not contain cliques of size more than $d + 1$. We immediately have that:

Lemma 3.20. *A potential-sink s in a d-degenerate graph has at most $d + 1$ undirected neighbors.*

Proof. Since s is a potential-sink, its undirected neighbors constitute a clique in the skeleton of the input graph. However, a d-degenerate graph can contain no clique that is larger than $d + 1$, as every induced subgraph has to contain a vertex of degree at most d. □

We are now able to give the proof of Theorem 1.1.

Proof of Theorem 1.1. Before we initialize the data structure, we compute a degeneracy ordering of the skeleton of the graph. This is possible in time $O(n + m)$ with the algorithm of Matula and Beck [1983]. Let the ordering be $v_1, v_2, \ldots, v_n$, i.e., v_i has at most d neighbors in $v_1, \ldots, v_{i-1}$. We iterate through the vertices and insert their incident edges to preceding neighbors. That is, if we handle a vertex v_i, we insert all edges $v_i - v_x$ and arcs $v_i \to v_y$, $v_z \to v_i$ with $x, y, z < i$. This way, v_i has degree at most d if we insert an edge for it and, thus, inserting the edge requires time $O(d)$, yielding an initial time of $O(dm)$.

We proceed as in the proof of Lemma 3.19 and maintain a stack of potential-sinks. While the stack is not empty, we use `pop-ps`(s) to orient edges towards s and remove s from the graph. Recall that this costs $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$. Removing a potential-sink s, by Lemma 3.20, produces costs of at most $O(d^2 + d \cdot \delta^-(s))$. Since this removes $\delta(s) + \delta^-(s)$ edges from the graph, we pay $O(dm)$ to remove *all* potential-sinks.

To obtain the corresponding extension, we remember the order in which we remove the potential-sinks. The reverse of that order is a topological ordering of an extension and, hence, can be used to extend G in linear time. □

Let us close this section by pointing out the advantage of an $O(dm)$ algorithm compared to an $O(\Delta m)$ algorithm. Many natural graph classes have bounded degeneracy,

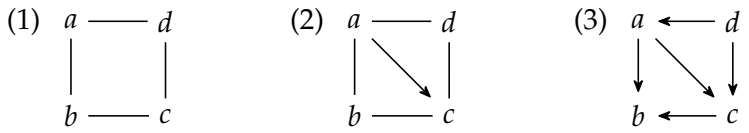


Figure 3.4: Graph (1) is not extendable: Orienting the cycle of length 4 will either create a v-structure or a directed cycle. (Intuitively, this explains why an undirected graph is extendable iff it is chordal.) However, the graph is an MPDAG by definition. Graph (2) on the other hand is extendable, a consistent DAG extension is shown in (3).

but unbounded maximum degree – for instance, planar graphs and graphs of bounded treewidth. Our algorithm runs in *linear time* on such graphs. Table 3.3 summarizes the findings of this section on some well-known graph classes.

3.4 A Graphical Characterization of Extendable MPDAGs

As discussed in the previous section, sink-based reduction algorithms can extend graphs in time $O(dm)$. The lower bounds suggests that no significantly faster algorithm is possible. However, the lower bounds do not rule out faster algorithms for more structured graphs. It is well-known that extensions of CPDAGs can be constructed in linear time by algorithms such as Maximum Cardinality Search [Tarjan and Yannakakis, 1984], as we discussed in Section 2.3.

A related graph class is the one of MPDAGs, which encode additional background knowledge compared to CPDAGs and are, in contrast to PDAGs, completed by the four Meek rules R1-R4 (Figure 3.1). Unlike CPDAGs, MPDAGs are not extendable by definition – see Figure 3.4. For practical use, however, only *extendable* MPDAGs are meaningful. Hence, it is our goal to provide a graphical characterization for such graphs. In addition, this characterization allows us to compute an extension in linear-time. Consider the graph (2) from Figure 3.4 and observe that in order to extend an MPDAG, it is not sufficient to orient the undirected subgraph (it is not necessarily chordal). In particular, when we study MPDAGs, we have to consider the directed edges, too. Therefore, we make use of a graphical object introduced by Perković [2020]:

Definition 3.21 (Bucket)**.** *Let G be an MPDAG and $C = (V_C, E_C, \emptyset)$ a connected component in $U(G)$. We call $B = G[V_C]$ a* bucket*.*

A bucket is the induced subgraph on the vertices of a connected component in $U(G)$ and, hence, may contain directed edges such as $a \to c$ in (2) of Figure 3.4.

Definition 3.22 (Bucket Graph)**.** *Let G be a graph. Then, $B(G)$ is the graph containing the vertices of G and all directed and undirected edges between vertices, which are in the same bucket in G. More formally: $B(G) = (V, E_G, A_{B(G)})$ with $A_{B(G)} = \{a \to b \in A_G \mid a - \cdots - b \in G\}$.*

Computationally, $B(G)$ can be constructed by starting with $U(G) = (V, E_G, \emptyset)$ and finding the connected components of it, and afterwards inserting arcs $a \to b \in A_G$ with a and b in the same connected component in $U(G)$. Based on this subgraph, we want to find a criterion for the extendability of MPDAGs. We derive the following statement, in analogy to the case for CPDAGs:

Lemma 3.23. *Let $G = (V, E_G, A_G)$ be an MPDAG. Let $C = (V, \emptyset, A_C)$ be a consistent extension of $B(G)$. Then, $D = (V, \emptyset, A_G \cup A_C)$ is a consistent extension of G.*

Proof. We show that by computing an AMO for every bucket we neither create a v-structure nor a cycle. The proof follows the general structure of the one of Proposition 2.15. Again, it is immediate that (1.) D is an orientation of G. It remains to show that (2.) it is acyclic and (3.) has the same v-structures as G.

For (3.), assume that D contains a v-structure $a \to b \leftarrow c$ that is not in G. Then, G contains either $a \to b - c$ or $a - b - c$ as induced subgraph. However, the first induced subgraph cannot occur in an MPDAG by definition (R1 would apply) and, in case of the second, $a \to b \leftarrow c$ cannot be in consistent extension C of B with the edges a, b, c being in one bucket.

For (2.), assume there is a directed cycle in D and consider the shortest one $c_1 \to c_2 \to \ldots \to c_p \to c_1$. As the cycle can neither be fully undirected nor directed in G (in the former case the whole cycle would be in one bucket leading to a similar contradiction as above), there has to be a $c_i - c_{i+1}$ with $c_{i-1} \to c_i$. For $p \geq 4$, R1 implies that c_{i-1} and c_{i+1} have to be adjacent for G to be an MPDAG. However, both $c_{i-1} \to c_{i+1}$ and $c_{i-1} \leftarrow c_{i+1}$ in D would lead to a shorter cycle. For $p = 3$, if one edge is undirected, then R2 would apply in G; if two edges are undirected, again the cycle is in one bucket. □

We investigate the structural properties of buckets in more detail.

Lemma 3.24. *Let G be an MPDAG. A bucket B of G does not contain a v-structure.*

Proof. Consider a v-structure $a \to b \leftarrow c$. For a, b, c to be in the same bucket, we assume w. l. o. g. that there is an undirected path between a and b not containing c: $b - p_1 - p_2 - \cdots - p_k - a$ with $k \geq 1$. Moreover, let a, b, c be chosen such that the path is shortest among all v-structures in the same bucket.

We have an edge between a and p_1 as well as c and p_1 (else G would not be an MPDAG due to R1). These edges cannot be directed $a \leftarrow p_1$ (or $p_1 \to b$, respectively) as otherwise R2 would apply. Moreover, the edges cannot both be undirected, as R3 would apply. Finally, setting one edge as $a \to p_1$ and the other as undirected would again imply R1 if a and c are nonadjacent (which they have to be to form a v-structure). Hence, we have $a \to p_1 \leftarrow c$. This excludes the case $k = 1$. For $k > 1$, p_1 would also be in the same bucket as a, c and the path from a to p_1 is shorter than to b. A contradiction. □

Hence, in order to determine whether an MPDAG G has a consistent extension, it suffices to determine whether $B(G)$ can be oriented acyclically and *morally*, that is without v-structure. In Chapter 2, we argued that an acyclic moral orientation of an undirected graph U (we used the term AMO in this context) exists if, and only if, U is chordal. Hence, for $B(G)$ to have a consistent extension it is necessary for its skeleton to be chordal by Corollary 2.18. The following result states that it is also sufficient. We note that preliminary results in this direction were obtained by Meek [1995] (Lemma 8). However, these are non-constructive and *assume* extendability. In contrast our proof shows both directions and is constructive, yielding a linear-time extension algorithm for MPDAGs.

Lemma 3.25. *Let G be an MPDAG. Graph $B(G)$ has a consistent extension iff its skeleton is chordal. Such a consistent extension can be computed in linear-time.*

Proof. We prove the first direction constructively with a linear-time algorithm that produces a consistent extension of $B(G)$ with a chordal skeleton.

The algorithm proceeds as follows: A Maximum Cardinality Search (MCS, see Section 2.3 for an introduction) is performed on the skeleton of $B(G)$. This produces the topological ordering τ of an acyclic and *moral* orientation of the skeleton in linear time as it is chordal by assumption (Proposition 2.21 and Lemma 2.17). By Lemma 3.24, we know that orienting $B(G)$ according to τ satisfies condition (3.) of a consistent extension (Definition 2.11) and condition (2.) is trivially satisfied. For satisfying (1.), for all directed edges $a \to b$ in $B(G)$, it has to hold that a comes before b in τ. To achieve this, the MCS is modified such that in its traversal, the algorithm always chooses a vertex, from the set of vertices with highest number of visited neighbors (which we denoted by M_i at step i), that has no incoming arcs in $B(G)$ from unvisited vertices (see function `bucket-mcs` in Algorithm 3.1). The corresponding set of vertices is denoted M_i^A, where A is the given set of arcs, which should be reproduced. Hence, the resulting ordering implies the directed edges in $B(G)$.

It remains to show that there is always such a vertex to choose, i.e., that M_i^A is non-empty. We prove by induction that, if G is an MPDAG, at every step of the algorithm, there exists a vertex with maximum number of visited neighbors, which has no incoming arc in $B(G)$. This is clear at the start (when every vertex has no visited neighbors and hence $M_0 = V$), as otherwise every vertex would have an incoming arc and, thus, we would have a cycle (every acyclic graph and every subgraph subgraph of it has at least one source).

Now assume the first i vertices have been visited and there was always a vertex with maximum number of visited neighbors without an incoming arc in $B(G)$. We show that there is again such a vertex. Note that not all vertices in M_i have an incoming arc from another vertex in this set (as this would imply a cycle in $B(G)[M_i]$). Hence, there has to be at least one vertex x in M_i with an incoming arc from an unvisited vertex not in M_i. Let $x \leftarrow y$ be such an incoming arc. As y has less visited neighbors than x (else it would also be in M_i), there has to be a previously visited neighbor z of x, which is not a neighbor of y. As the algorithm has visited z before x we have:

1. either B contains the arc $z \to x$, which would be a contradiction by Lemma 3.24 as this would be a v-structure in a bucket

2. or B contains the edge $z - x$, which would imply the induced subgraph $z - x \leftarrow y$ (and this is also induced in G by the construction of $B(G)$) contradicting the fact that G is an MPDAG.

By induction hypothesis, the arc $z \leftarrow x$ cannot be in B. This proves that M_i^A is not empty and hence the correctness of the algorithm.

Clearly, the algorithm can be implemented in linear time. For the adapted MCS, note that similar techniques as in the standard MCS can be used, just with sets of vertices split into two parts, the vertices that have no incoming arc from an unvisited vertex and the remaining ones. These parts can be efficiently maintained, for example by having a counter of incoming arcs for each vertex.

For the reverse direction, note that for a non-chordal skeleton there cannot exist an orientation, which is acyclic and moral, see Corollary 2.18. □

Hence, we obtain the following graphical characterization of extendable MPDAGs – and a linear-time algorithm for computing extensions.

Theorem 3.26. *An MPDAG is extendable iff the skeleton of every bucket is chordal.*

Proof. Combine Lemma 3.23 and Lemma 3.25. □

Algorithm 3.1: Computing a consistent extension of an MPDAG in linear time.

input : An MPDAG $G = (V, E, A)$.
output: Consistent extension of G.

function `bucket-mcs(`U, A`)`
  $\tau_0 :=$ empty list
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    `// Similar to Algorithm 2.2, the next three lines can be implemented efficiently with appropriate data structures`
    foreach $v \in V$ **do** $P_i(v) := Ne_U(v) \cap \tau_i$
    $M_i := \{v \in V \setminus \tau_i \mid \text{for all } w \in V \setminus \tau_i \text{ it holds that } |P_i(v)| \geq |P_i(w)|\}$
    $M_i^A := \{v \in M_i \mid \text{ there exists no } w \in V \setminus \tau_i \text{ such that } w \to v \in A\}$
    $x :=$ any element of M_i^A
    $\tau_{i+1} := \tau_i + (x)$ `// append` x `to` τ_i
  end
  return τ_n
end
`// Compute` $B(G)$ `as defined in Definition 2.11.`
$B(G) := (V, E_G, A_{B(G)})$ with $A_{B(G)} = \{(a, b) \in A_G \mid a - \cdots - b \in G\}$
$\tau :=$ `bucket-mcs(skel(`$B(G)$`)),` $A_{B(G)}$`)`
$D :=$ `orient(`G, τ`)`
return D

Theorem 3.27. MPDAG-EXT *can be solved in linear time* $O(n + m)$.

Proof. Perform the modified MCS from the proof of Lemma 3.25 on $B(G)$. Afterwards, test in linear time whether the orientation is moral [Tarjan and Yannakakis, 1984]. The first part of this approach is described in Algorithm 3.1. □

3.5 Recognition of Causal Graph Classes

As shown in the previous section, it is possible to find a consistent extension of an MPDAG in linear-time. The remaining question is how to check whether a partially directed graph is an MPDAG. Checking acyclicity can be done in linear time, but testing the closedness under the Meek rules R1-R4 is harder as the lower bound in Section 3.2 suggests.

In this section, we develop an algorithm that outperforms a naive detection of the Meek rules – which alone for the rules R3 and R4 requires time $O(\Delta^3 n)$ or $O(\Delta^2 m)$. Our approach runs in time $O(\Delta m)$ and, thus, matches the $O(n^3)$ lower bound that is prescribed by the Strong Triangle Conjecture.

Theorem 3.28. MPDAG-REC *can be solved in time* $O(\Delta m)$.

Proof. The algorithm checks step-by-step whether the MPDAG criteria are satisfied:

1. If G contains a directed cycle, return "No".
2. R1: If $\exists\, b - c$ and vertex a s.t. $a \to b - c$, return "No".
3. R2: If $\exists\, a - c$ and vertex b s.t. $a \to b \to c$, return "No".

4. R3: If $\exists\, d \to c$ and vertex a s.t. $a \underbrace{\text{—}\, d \to}\, c$ and vertex b s.t. $b \to c \leftarrow d$, return "No".
5. R4: If $\exists\, c \to b$ and vertex d s.t. $d \to c \to b$ and a s.t. $a \underbrace{\text{—}\, b \leftarrow}\, c$, return "No".
6. Return "Yes".

The rules R1 and R2 are detected naively. Hence, it is clear that the subgraphs corresponding to R1 or R2 cannot be present in the graph, when the algorithm reaches R3 and R4. For the detection of these rules, the existence of vertices a and b in R3 and d and a in R4 is checked separately. Therefore, it remains to show that, indeed, it is not necessary to explicitly check whether there is an undirected edge $a - b$ in R3 or $d - a$ in R4.

Consider R3. If (i) a and b were nonadjacent, R1 would apply to $b \to c - a$; if (ii) $a \to b$, R2 would apply to $a \underbrace{\to b \to}\, c$; if (iii) $a \leftarrow b$, R1 would apply to $b \to a - d$. Hence, we need to have $a - b$. For R4, if (i) d and a were nonadjacent, R1 would apply to $d \to c - a$; if (ii) $a \to d$, R2 would apply to $a \underbrace{\to d \to}\, c$; if (iii) $a \leftarrow d$, R1 would apply to $d \to a - b$. Hence, we need to have $d - a$.

It follows that the algorithm outputs "Yes" if, and only if, no cycle is present and no rule from R1-R4 applies, i. .e., when the graph is an MPDAG. The run time follows immediately, as each edge is examined a constant number of times and only neighboring vertices are considered. □

We conclude that it is possible to speed up the detection of R3 and R4 by not searching for 4-tuples (a, b, c, d), but, e.g. in the case of R3 for 3-tuples (a, c, d) and (b, c, d).

We summarize the complexity of the recognition problem for the remaining graph classes. The result for PDAGs is straightforward and we list it here for the sake of completeness. To the best of our knowledge, the results for CPDAGs and CGs have not been stated previously.

Proposition 3.29. *The problems* PDAG-REC, CPDAG-REC, *and* CG-REC *can be solved in time* $O(n + m)$.

Proof. For PDAG-REC, we only have to check that the input does not contain a directed cycle. For CPDAG-REC the algorithm proceeds as follows (the input is a partially directed graph G):

1. Find a consistent extension of G by using Algorithm 2.1.
2. If D contains a cycle, output "No". Otherwise find the CPDAG corresponding to D (let this be C) and output "Yes" if $C = G$ and "No" otherwise.

Step 2 can be performed in $O(n + m)$ as finding the CPDAG representing the Markov equivalence class of a DAG is possible in expected linear time [Chickering, 1995]. Hence, the whole algorithm runs in time $O(n + m)$. If G is a CPDAG, then D is a consistent extension by Theorem 2.1. Hence $C = G$. If G is not a CPDAG, either D contains a cycle or the graph C is computed, which is by definition a CPDAG. Hence, $C \neq G$.

Finally, chain graphs can be recognized in linear time as well. We use the following contraction based algorithm: As long as the input graph contains an undirected edge, we contract it to an arbitrary neighbor. Once all undirected edges are gone, we simply check whether the remaining graph is acyclic.

If the input contains a semi-directed cycle, the undirected parts of the cycle get contracted and the resulting graph contains a directed cycle. On the other hand, any directed cycle in the contracted graph corresponds to a cycle in the original graph with at least one directed edge. □

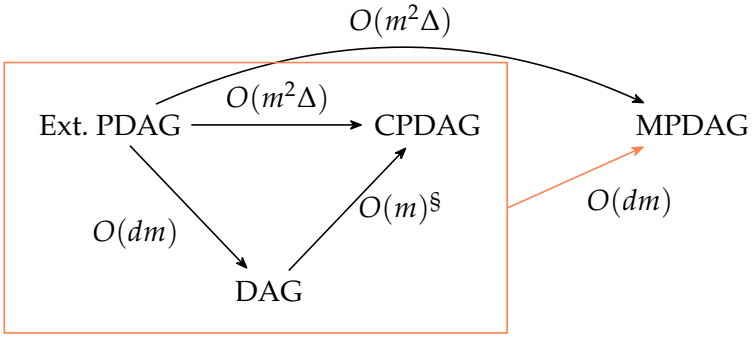


Figure 3.5: Using the techniques from Section 3.5, it follows that (extendable) PDAGs can be maximally oriented into CPDAGs/MPDAGs in $O(m^2\Delta)$ by direct application of the Meek rules. Using the new PDAG extension algorithm from Section 3.3, PDAG-to-CPDAG can be performed in expected time $O(dm)$ as DAG-to-CPDAG is possible in time $O(m)$ (§: [Chickering, 1995]). Below, we give an $O(dm)$ algorithm for PDAG-to-MPDAG, which uses a consistent DAG and the corresponding CPDAG as auxiliary graphs.

3.6 Application to Maximal Orientations of PDAGs

In this section, we study the problem MAXIMALLY-ORIENT, i.e., the task of computing the closure of a PDAG under the orientation rules R1-R4. As we noted in Section 3.4, for applications in causality, only *extendable* models are meaningful. In particular, a non-extendable PDAG has no causal explanation and closing such a graph under the orientation rules R1-R4 is purposeless. This justifies that our algorithm for MAXIMALLY-ORIENT assumes that the given PDAG is extendable. We will see that the ideas of the previous sections can be combined to obtain more efficient algorithms for this problem. An overview of the results is given in Figure 3.5.

In general, MAXIMALLY-ORIENT produces an MPDAG when the input is an extendable PDAG. If the directed edges follow from v-structure information, i.e, there is no additional background knowledge, the resulting graph is a CPDAG. In this setting, Chickering [2002a] noticed that, in order to maximally orient a PDAG into a CPDAG, it is not necessary to apply the Meek rules directly. A more efficient computation is possible by first extending the PDAG into a DAG and afterwards transforming the DAG into the corresponding CPDAG. The latter task can be performed in linear time [Chickering, 1995]. In combination with the PDAG extension algorithm presented in this work, we can conclude that PDAG-to-CPDAG can be performed in time $O(dm)$ via the route PDAG-DAG-CPDAG. Notably, this is significantly faster than directly applying the Meek rules, which takes time $O(m^2\Delta)$ (with the improvements for detecting the Meek rules that we discussed in the previous section).

Building on this, we want to solve the general problem MAXIMALLY-ORIENT for extendable graphs. Here, the issue is that a DAG-to-MPDAG routine is not possible, as a DAG does not correspond to a unique MPDAG. Hence, we have to include the edge information from the PDAG to perform such a step. The idea is to utilize a topological ordering induced by a consistent extension D, in order to traverse the PDAG only once while detecting the Meek rules. This is based on the observation that an orientation $u - v$ into $u \to v$ through R1-R4 only relies on parents of v in any consistent extension. In this way, it is possible to obtain a run time $O(\Delta m)$. By additionally making use of the CPDAG corresponding to D, we can improve even further:

Algorithm 3.2: Maximal orientation of an extendable PDAG.

input : Extendable PDAG G.
output: Maximal orientation of G.

$D :=$ consistent extension of G
$\tau :=$ topological ordering of D
// $\tau^{-1}(v)$ denotes the position of v in τ
$C :=$ CPDAG of D
// the following modifications are performed in place in C
foreach *connected component U of the undirected subgraph of C* **do**
  Copy edge directions from G to U
  foreach $v \in V_u$ *in order* τ **do**
    // R1
    foreach $p - v$ *with* $\tau^{-1}(p) < \tau^{-1}(v)$ **do**
      if $\exists a$ *s.t.* $a \to p - v$ **then** Orient $p \to v$.
    end
    // R4
    foreach $p \to v$ *with* $\tau^{-1}(p) < \tau^{-1}(v)$ **do**
      if $\exists d, a$ *s.t.* $d \to p \to v$ *and* $a \overset{\frown}{-} v \overset{\frown}{\leftarrow} p$ **then** Orient $a \to v$.
    end
    // R2
    foreach $p - v$ *with* $\tau^{-1}(p) < \tau^{-1}(v)$ *in decreasing* $\tau^{-1}(p)$ **do**
      if $\exists b$ *s.t.* $p \overset{\frown}{\to} b \to v$ **then** Orient $p \to v$.
    end
  end
end
return C

Theorem 3.30. *For extendable PDAGs, problem* MAXIMALLY-ORIENT *can be solved in time* $O(dm)$.

Proof. MAXIMALLY-ORIENT can be solved with Algorithm 3.2. It applies the Meek rules starting with CPDAG C instead of PDAG G. Such an approach is valid, see for example the proof of Theorem 4 in Meek [1995].

Moreover, it is sufficient to apply the Meek rules inside the connected components of $U(C)$ (with the directed edges from G copied over). This is because a vertex outside these connected components cannot take part in the rules R1-R4. The only such vertices could potentially be a in R1 and b in R2, but clearly the rules cannot be applied in C immediately and directing further edges will not create such a situation. Therefore, we obtain the following problem. Given a PDAG H with chordal skeleton and no v-structures, and a topological ordering τ of a consistent extension, exhaustively apply R1-R4. Note that R3 cannot occur as it contains a v-structure.

The algorithm visits the vertices in topological order. We prove by induction that after handling vertex v, the induced subgraph on all visited vertices coincides with this induced subgraph in M. After visiting the first vertex, the proposition above holds as the induced subgraph contains no edge. Now assume that the statement holds until the vertex u is visited immediately before v. Then, all edges going into any *visited* vertex except v have already been set correctly by induction hypothesis. To complete the proof,

we only have to show that every edge into v is correctly oriented or left undirected. Note that in case we direct the edge, only the orientation consistent with τ is valid.

It is easy to see that it is sufficient to consider vertices coming before v in τ for Meek rule detection. The correctness of the fast Meek rule detection was already argued in the proof of Theorem 3.28. However, we have to be careful due to the fact that applying a rule to some edge $p_1 - v$ may lead to the orientation of $p_2 - v$ at some later point. Hence, we have to make sure that edges $p_2 - v$ do not have to be rechecked for applicability of R1-R4 repeatedly.

We can see that rechecking for R1 is not necessary as the edge incident to v has to be *undirected* (vertex v corresponds to vertex c in R1, as we want to find new directed edges into v). In R4 (here v corresponds to vertex b) there is such a directed edge $c \to b$ incident to v, but this edge always follows from R1 (because of $d \to c$). Hence, by first exhaustively applying R1 and afterwards R4, these rules are handled correctly. As R3 does not apply, R2 remains. Here, we consider the edges $p - v$ with decreasing $\tau^{-1}(p)$, i.e., the closest p first. When considering p, we hence know that for all p' with $\tau^{-1}(p) < \tau^{-1}(p') < \tau^{-1}(v)$, R2 has already been applied. Thus, when searching for $p \to p' \to v$ we know that $p' \to v$ has already been directed if it is directed in M.

It is immediately clear that the algorithm runs in time $O(\Delta m)$. To see that it is actually $O(dm)$, observe that we only consider parents of vertices in D. As we are in an undirected component without any v-structures, the parents have to be fully connected, i.e., form a clique together with v. But d-degenerate graphs contain cliques with at most $d + 1$ vertices. □

3.7 Conclusions

We proposed a method of time complexity $O(n^3)$ to find a consistent DAG extension in a given PDAG that improves upon the commonly used algorithm by Dor and Tarsi [1992]. It is based on a new data structure for potential-sinks in PDAGs, which makes it easily implementable and practically useful. By applying a fine-grained complexity analysis, we showed that our algorithm is optimal under the Strong Triangle Conjecture. This answers the open question on the existence of a linear-time method for the extension problem in PDAGs negatively conditional on this conjecture. Through a refined analysis, we showed that our algorithm runs in linear time on practically important graphical causal models, such as graphs with bounded treewidth. Based on these results, we provided a precise complexity-theoretic classification of the extension problem, also including more structured graph classes. As part of this, we gave a graphical characterization of extendable MPDAGs: a result, which moreover yields a linear-time extension algorithm for this graph class. Finally, we applied the new methods to the corresponding recognition problems and developed the techniques further to design an effective algorithm for closing a PDAG under the orientation rules of Meek.

4

Enumerating Markov Equivalent DAGs

The previous chapter discussed the problem of testing extendability of causal graphical models, that is whether a graph with directed *and* undirected edges admits a causal explanation. The algorithms were constructive in the sense that they return an extension if it exists. However, they only give a single arbitrary representative of the set of consistent extensions of the given graph.

In this chapter, we build on these tools in order to deal with the problem of *enumerating the DAGs in an MEC* given its CPDAG (we actually solve the more general problem of enumerating all consistent extensions of a PDAG), that is listing each member of the MEC exactly once (see Figure 4.1 as an example of an MEC and its CPDAG). This task is an important primitive in causal analysis, used as a subroutine to solve more complex problems in software packages such as `pcalg` [Kalisch et al., 2012], `causaldag` [Squires, 2018], `TETRAD` [Ramsey et al., 2018] and `dagitty` [Textor et al., 2016]. Enumeration of an MEC's members can be applied to solve many important downstream tasks in causal inference. For example, one can estimate the causal effect of the exposure variable on the outcome for each DAG in the equivalence class, which is learned from the observed data [Maathuis et al., 2009]. One could also check for every DAG whether it conforms to additional domain information or background knowledge in order to find the most plausible DAG [Meek, 1995], or select intervention targets to distinguish between certain DAGs in the equivalence class [He and Geng, 2008, Hauser and Bühlmann, 2012]. While there are custom algorithms, which avoid the use of explicit enumeration (sometimes by settling for approximate solutions), for many of these cases, it remains a flexible and very general tool that can be utilized even when other methods fail. The main drawback is, of course, its high computational cost, which we aim to address in this chapter.

Any method for enumerating the DAGs in an MEC requires exponential time in the worst-case, due to the basic fact that there are classes with exponential size. A crucial feature from a computational perspective is thus the *delay:* the algorithm's run-time between two consecutive output DAGs, which should be as small as possible. Another desirable property would be that subsequent DAGs smoothly change their structure, i.e., share most of their edge orientations, which constitutes a more plausible enumeration from the causal point of view. In this chapter, we take both these aspects into account.

To the best of our knowledge, no study has been published that performs a systematic analysis of the enumeration problem, including its algorithmic aspects. One commonly used folklore algorithm is based on the rules proposed by Meek [1995], which we introduced in the previous chapter (Figure 3.1), to transform a causal graph (e.g., a CPDAG or PDAG) into its maximal orientation. Such an approach is used for example in the `TETRAD` package. Applying the Meek rules has the property that any remaining

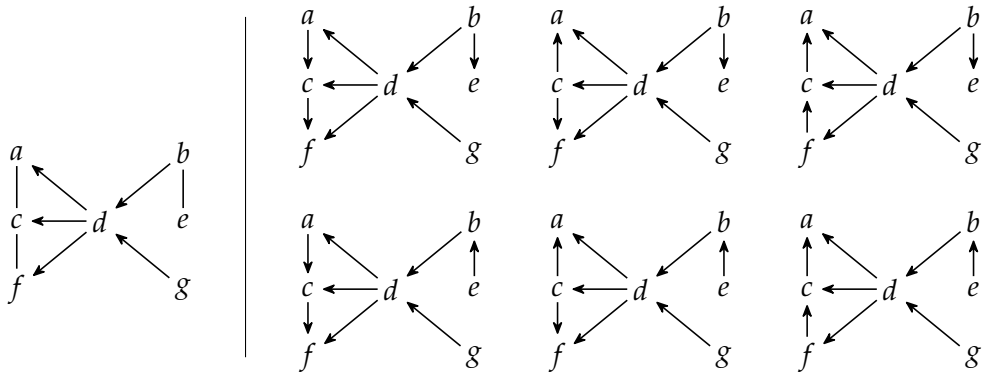


Figure 4.1: An MEC on the right, which consists of six DAGs. This class is represented by the left CPDAG, which uniquely represents the MEC by including undirected edges if two DAGs differ in their direction.

undirected edge $a - b$ is oriented $a \to b$ in at least one and $a \leftarrow b$ in another DAG represented by the graph. Consequently, the DAGs can be enumerated by successively trying both possible orientations. This yields a polynomial delay algorithm, but the degree of the corresponding polynomial is rather large since the Meek rules have to be applied at *every* step. Another folklore approach is based upon the *transformational characterization* of MECs given by Chickering [1995], which states that two DAGs in the same MEC can be transformed into each other by successive single-edge reversals. This approach is used in the `causaldag` package. Hence, the MEC can be explored through such edge reversals starting from an arbitrary DAG in the class. The issue with this method is that already output DAGs need to be stored and every time a new DAG is explored it has to be checked that it has not been output before. This leads to a relatively large delay and memory demand. As both algorithms (which we call MEEK-ENUM and CHICKERING-ENUM) have not been explicitly stated in a publication, we give a formal description of both in Section 4.1 as well as a rigorous analysis of their delay.[1]

The main contribution of this chapter is the first $O(n + m)$-delay algorithm that, for a given CPDAG representing an MEC, lists all members of the class. We also show that the algorithm can be generalized to enumerate DAGs represented by a PDAG or MPDAG – causal models incorporating background knowledge. To achieve these results, we utilize Maximum Cardinality Search (MCS) [Tarjan and Yannakakis, 1984] originating from the chordal graph literature. In addition to the theoretical results, we give an efficient practical implementation, which is significantly faster than implementations of MEEK-ENUM and CHICKERING-ENUM.

We also propose a complementary method with the property that during enumeration subsequent DAGs gradually change their structure. This method utilizes the results by Chickering [1995], but performs the traversal of the MEC in a more refined way. Using such an approach, it is possible to output all Markov equivalent DAGs in sequence with the property that two successive DAGs have *structural Hamming distance* (SHD) at most three. This result is tight in the sense that there are MECs whose members cannot be enumerated in a sequence with maximal distance at most two. We also show that this observation can be used in the more general setting of enumerating maximal ancestral

[1]The libraries `pcalg` and `dagitty` use neither of the two approaches described here. We will not discuss these implementations further, but mention that they do not have polynomial-time delay.

graphs (MAGs) which encode conditional independence relations in DAG models with latent variables [Richardson and Spirtes, 2002].

4.1 Background

Enumerating (also known as *generating* or *listing*) all configurations satisfying a given specification is a fundamental task in combinatorics with classic problems being the enumeration of all permutations of n objects or all combinations of k of them. From a computational point of view, the main interest is to enumerate such objects as efficiently as possible. In this work we are interested in enumerating orientations of graphs, that is the input is a graph and the output is a list of oriented graphs satisfying certain properties. There are different ways of classifying algorithms for this task, respectively measuring their efficiency, see e.g. the classification given by Johnson et al. [1988].

Measuring the time complexity in the size of the input is often not desirable as the output size is commonly exponentially large, making a classical complexity analysis hard to interpret. A simple adaptation is to measure the time complexity in terms of combined input *and* output size. If the complexity of an algorithm can be expressed by a polynomial in this way, it is called a *polynomial total time* algorithm. However, this is still not a very fine measure of complexity, thus necessitating more restrictive definitions of "efficient" enumeration approaches. A popular and useful concept is that of *delay*, that is the maximum time that passes between two consecutive outputs. This complexity is measured in terms of the input size but can still be polynomial even for exponentially large outputs, which makes it easy to interpret. Delay algorithms have the crucial practical advantage that outputs are generated on the fly, which allows working with the first configurations without waiting for all to be generated.

When listing configurations one-by-one, it is moreover desirable to use only polynomial space during generation, thus in particular not storing all past configurations. This is an important feature in practice as it avoids memory problems in case of exponentially large outputs. Generally, we aim to give enumeration algorithms which have polynomial (or better *linear*) delay and space requirements. In particular, we are mainly interested in the following enumeration problem

Problem 4.1. ENUM-EXT

Instance: *A PDAG* $G = (V, E, A)$.
Result: *Listing of all consistent extensions* $EXT(G)$, *each exactly once.*

Another aspect is to generate enumeration sequences with successive configurations being "close" to each other. Closeness is formalized in this context such that the configurations can be transformed into each other through few (a constant amount of) changes. This is also known as Gray code enumeration and it has a rich literature in combinatorics [Savage, 1997, Mütze, 2022, Knuth, 2014].

Before we tackle Problem 4.1 from these perspectives, we give a formal description of the folklore algorithms based on [Meek, 1995] and [Chickering, 1995] as those approaches have never been explicitly discussed, in particular, with regard to their delay.

Enumeration based on Meek's rules. As presented in Figure 3.1 in the previous chapter, Meek [1995] gave a complete set of four rules (R1 - R4), which, when applied repeatedly, transform a PDAG into its maximal orientation (an MPDAG), i.e., orient all undirected edges, which are fixed in the consistent extensions of the PDAG.

A graph completed under the Meek rules has the property that every undirected edge $x - y$ is *not* fixed in the DAGs it represents, i.e., there is one which contains $x \rightarrow y$ and one which contains $x \leftarrow y$. Thus, the consistent extensions can be enumerated by first orienting the edge as $x \rightarrow y$ (and recursively orienting the remaining edges one-by-one, always after completing the graph under the Meek rules to ensure that every orientation yields a valid DAG) and afterwards orienting the edge as $x \leftarrow y$. We call this approach MEEK-ENUM and it is stated in Algorithm 4.1.

Algorithm 4.1: Enumeration algorithm of Markov equivalent DAGs based on Meek's rules (MEEK-ENUM).

input : An extendable PDAG $G = (V, E, A)$.
output: All consistent extensions of G.

```
1  function enumerate(G)
2      H := maximal orientation of G
3      if E_H is empty then          // E_H is the set of undirected edges of H
4          Output H
5      else
6          u − v := any element from E_H
7          Orient u − v into u → v in H
8          enumerate(H)
9          Reverse the orientation u → v into u ← v in H
10         enumerate(H)
11         Unorient u ← v into u − v in H
12     end
13 end
14 enumerate(G)
```

MEEK-ENUM has polynomial-delay, particularly as every orientation leads to a valid DAG (there are no "dead-ends" in the recursive search). However, it requires the application of Meek's rules in every step, which, by Theorem 3.30 of the previous section, is possible in time $O(n^3)$ (a "naive" application of the Meek rules takes significantly longer, amounting to a polynomial run-time of high-degree). This yields the following bound on the delay.

Proposition 4.2. MEEK-ENUM *(Algorithm 4.1) can be implemented such that all consistent extensions are enumerated with worst-case delay* $O(m \cdot n^3)$.

Proof. Clearly, every consistent extension is output as every undirected edge is oriented in both directions (by soundness of the Meek rules every directed edge is actually fixed in the DAGs in the class). No consistent extension is output twice because in each step, an undirected edge is oriented and fixed, i.e., every recursive call receives a different graph as input. Nor is an invalid DAG output, because every undirected edge can be oriented in both directions leading to a valid DAG by completeness of the Meek rules. In the worst case, the algorithm needs m recursive calls until a DAG is output, and a single call costs time $O(n^3)$ due to the application of Meek's rules by Theorem 3.30. Hence, the delay between two outputs is bounded by $O(m \cdot n^3)$. □

Enumeration based on [Chickering, 1995]. Another approach for enumerating Markov equivalent DAGs is build on the transformational characterization of MECs by Chickering [1995]. It is based on the notion of *covered edges*:

Definition 4.3 (Covered edge [Chickering, 1995])**.** *An edge $x \to y$ is called* covered *if* $Pa(x) \cup \{x\} = Pa(y)$.

Theorem 4.4 (Chickering [1995])**.** *Let D and D' be two Markov equivalent DAGs. There exists a sequence of* $\mathrm{SHD}(D, D')$ *edge reversals in D with the following properties:*

1. *Each edge reversed in D is a covered edge.*
2. *After each reversal, D is a DAG Markov equivalent to D'.*
3. *After all reversals, $D = D'$.*

Here, $\mathrm{SHD}(D, D')$ refers to the *structural Hamming distance* between D and D', that is the number of vertex pairs which have differing edge relations in D and D'. Theorem 4.4 states that starting from a DAG D we reach all DAGs in the same Markov equivalence class by reversals of covered edges. This permits the following enumeration approach, which we call CHICKERING-ENUM.

Algorithm 4.2: Enumeration algorithm of Markov equivalent DAGs based on Chickering's characterization [Chickering, 1995] (CHICKERING-ENUM).

input : A CPDAG $G = (V, E, A)$.
output: $[G]$.

```
1  function enumerate(D, vis)
2      Output D
3      foreach DAG D' obtained by reversing a covered edge in D do
4          if D' ∉ vis then
5              vis := vis ∪ {D'}
6              enumerate(D', vis)
7          end
8      end
9  end
10 D := any DAG in [G]
11 vis := {D}
12 enumerate(D)
```

Starting from D, similar to a depth-first-search (DFS), all neighbors of D (graphs with a single reversed covered edge) are explored and this is continued recursively. Eventually all Markov equivalent DAGs are reached by Theorem 4.4 above. In order to not visit any DAG twice, it is necessary to store a set of all visited DAGs. This is a major disadvantage of this algorithm leading to worst-case exponential space requirements.

Proposition 4.5. CHICKERING-ENUM *(Algorithm 4.2) enumerates a Markov equivalence class and can be implemented with worst-case delay $O(m^3)$.*

Proof. Clearly, any DAG is output exactly once by Theorem 4.4 and storing all visited DAGs ensures not outputting any DAG multiple times.

For the delay, it is first useful to analyze when the most steps between two outputs are necessary. This happens when the algorithm traverses back up the recursion tree (from a leaf), potentially up to the root, without outputting any new DAG and then down into another subtree. In principle, the recursion depth might be exponential and this would, in turn, lead to an exponential-time delay.

However, it is possible to bound the recursion depth by m (by Theorem 4.4 every edge is reversed at most once) and one will still find all DAGs in the MEC. It remains to estimate the cost at each recursion step. Going up the recursion tree, at each DAG, $O(m)$ neighbors might be checked whether they have been visited before, e.g., if they are in vis, and all of them might indeed be. The check takes time $\Omega(m)$ per neighboring DAG D' as the whole DAG has to be "read" at least once for lookup. If one uses a hash table for storing the DAGs, expected time $O(m)$ can be reached.

So, in total, we have $O(m)$ recursion steps between two outputs, with at most $O(m)$ neighboring DAGs being considered, each of which consuming $O(m)$ time for lookup in vis. This leads to a worst-case delay of $O(m^3)$. □

The algorithm is stated for CPDAGs, however, it is rather easy to see that it also works for listing all consistent extensions of PDAGs with only minor modifications.

4.2 Enumerating the Members of an MEC

The input for the considered enumeration problem stated in Problem 4.1 is a PDAG. However, to make things easier, we will at first restrict ourselves to CPDAGs, i.e., a more structured subclass of PDAGs representing an MEC. Recall Section 2.3 for a detailed discussion on how to find *any* DAG in the MEC represented by a CPDAG in linear-time. Proposition 2.15 showed that every consistent extension of CPDAG G corresponds to an AMO of its undirected subgraph $U(G)$. Hence, the problem immediately reduces to the task of enumerating the AMOs of a chordal graph.

In Algorithm 2.2, we presented the well-known MCS algorithm, which is adapted to yield an AMO provided it is run on a chordal graph, such as $U(G)$. We now show that this algorithm is, in principle, able to produce any AMO of $U(G)$ and, consequently, any consistent extension of CPDAG G.

Definition 4.6 (MCS ordering)**.** *Let $G = (V, E)$ be an undirected graph. A linear ordering $\tau = (v_1, \dots, v_n)$ is an* MCS ordering *if it is the visit order of an MCS, that is for each v_i, it holds that $Ne(v_i) \cap \{v_1, \dots, v_{i-1}\}$ is maximum over $v_i, \dots, v_n$.*

This is equivalent to v_i being in M_{i-1} as defined in Section 2.3. Not every PEO of G is an MCS ordering, however, they are still expressive enough to represent all AMOs of a chordal graph.

Lemma 4.7. *Let $G = (V, E)$ be a chordal graph. Every AMO D of G can be obtained by orienting the edges according to an MCS ordering τ, i.e. as $D = G[\tau]$.*

Proof. We give a constructive proof. Consider a modified version of MCS, which always picks a vertex $x \in M_i$ such that there is no $w \to x$ for a $w \in V \setminus \tau_i$ in D. Clearly, this MCS produces a topological ordering inducing D and this ordering is an MCS ordering.

It remains to show that such a vertex $x \in M_i$ always exists. Assume, for the sake of contradiction, that this is not the case. Then, all vertices x in M_i have at least one incoming edge $w \to x$ from a $w \in V \setminus \tau_i$. For at least one x, there exists a w with $w \to x$ which is not in M_i. Else D would contain a directed cycle as the induced subgraph $D[M]$

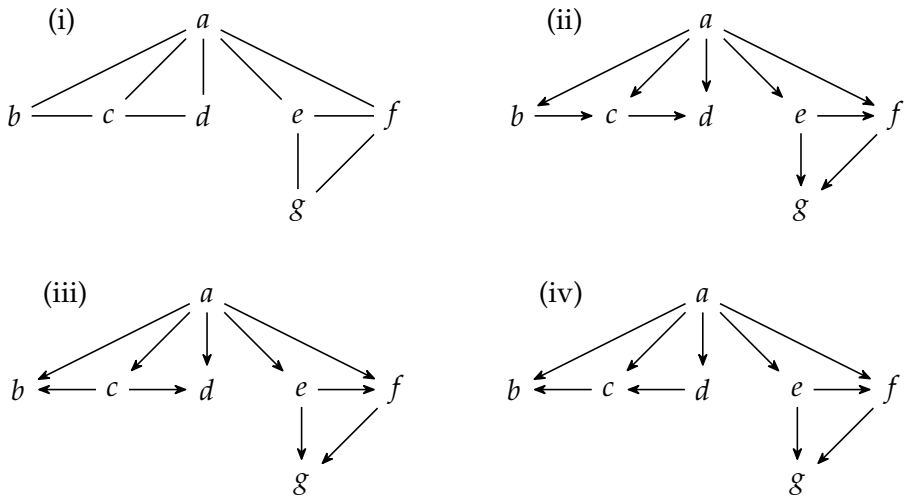


Figure 4.2: An example showing for the chordal graph (i) that, at a given step i of the algorithm, not all vertices in M_i can be chosen. If the MCS starts with vertex a, all neighbors b, c, d, e, f have the same number of visited neighbors, namely 1. While an MCS may choose any one of them, we *cannot* choose *all* one-after-the-other in our enumeration. Choosing b or e as the second vertex may yield the same AMO as (a, b, c, d, e, f, g) and (a, e, f, g, b, c, d) are both topological orderings of (ii). However, choosing vertices in M_i from one connected component in $G[\{b, c, d, e, f, g\}]$ such as $\{b, c, d\}$ one-after-another will yield distinct AMOs (in (iii) and (iv) AMOs with c, d chosen as second vertex are given).

would not have a source vertex. However, with w not in M_i, there exists, by definition, a vertex $v \in \tau$, which is adjacent to x but not w. This implies $v \to x \leftarrow w$ in D, which contradicts the morality of D. □

This observation yields a new enumeration approach: Instead of producing a single AMO by choosing an arbitrary vertex in each step of the MCS, perform multiple choices and recur for each of them – eventually listing all AMOs by Lemma 4.7 and thus generating all consistent extensions of a CPDAG by Proposition 2.15 and Corollary 2.19. There is one pitfall however: We *cannot* simply choose *every* vertex from M_i one after the other as some graphs would be output multiple times. Figure 4.2 illustrates this issue: If vertex a has been visited, the next vertex could be b, c, d, e, or f (all have one visited neighbor, namely a). But choosing, say, b or e may lead to the same AMO as the order of b and e after choosing a is irrelevant.

This issue can be addressed as follows: While the choice of the first vertex v from M_i is arbitrary, every other vertex x has to be connected to v in $G[V \setminus \tau]$. If they are connected, then the order of v and x matters. Otherwise, choosing x instead of v would lead to duplicate AMOs being output. In our example, this means that if b is the first considered vertex in M_i after a has been visited, the other choices we would consider are c and d as these are the vertices reachable from b in the induced subgraph over the unvisited vertices.

Lemma 4.8. *Let $G = (V, E)$ be a chordal graph and τ_i the visit sequence of an MCS after i steps with $i < n$.*

1. *If $x, y \in M_i$ are connected in $G[V \setminus \tau_i]$, the set of AMOs produced by visiting x next is disjoint from the set produced by choosing y next.*

2. *If $x, y \in M_i$ are unconnected in $G[V \setminus \tau_i]$, any AMO produced by visiting y as the next vertex can be produced by choosing a vertex in M_i connected to x in $G[V \setminus \tau_i]$ next.*

Proof. We show item (1.) first and let p be the shortest path between x and y in $G[V \setminus \tau_i]$. Since AMOs do not contain v-structures, any AMO choosing x next must orient p as $x \rightarrow \ldots \rightarrow y$ while any AMO with y next yields $x \leftarrow \ldots \leftarrow y$.

For item (2.) let $C_1, \ldots, C_k$ be the connected components of $G[V \setminus \tau_i]$ with $x \in C_1$. Any topological ordering with prefix τ_i can be rewritten as $\tau_i, C_{\pi(1)}, \ldots, C_{\pi(k)}$ for an arbitrary permutation π – in particular for $\pi = \mathrm{id}$. □

The following Lemma provides a simplified and more efficient way of testing whether two vertices in M_i are connected.

Lemma 4.9. *Let $G = (V, E)$ be a chordal graph and τ_i the visit sequence of an MCS after i steps with $i < n$. It holds that vertices $x, y \in M_i$ are connected in induced subgraph $G[V \setminus \tau_i]$ iff they are connected in $G[M_i]$.*

Proof. Trivially, if x and y are connected in $G[M_i]$ so they are in $G[V \setminus \tau_i]$. Hence, we only have to prove that if they are not connected in $G[M_i]$, then they are also disconnected in $G[V \setminus \tau_i]$. The base case of τ_i being empty is trivial as then we have $M_i = V \setminus \tau_i$. So let τ_i be non-empty and consider two arbitrary vertices $x, y \in M_i$ that are in different connected components of $G[M_i]$. For a contradiction assume that they are connected in $G[V \setminus \tau_i]$ via a shortest path $\pi = x - p_1 - \cdots - p_\ell - y$ with $\ell \geq 1$ and $p_1, \ldots, p_\ell \in V \setminus \tau_i$.

On π, there has to be a vertex $p_k \notin M_i$, else x and y are connected in $G[M_i]$. Hence, there is (i) a vertex $z \in P_i(x)$, which is not in $P_i(p_k)$, and (ii) a vertex $z' \in P_i(y)$, which is not in $P_i(p_k)$ by definition of M_i. We first consider the case that z or z' is a common neighbor of both x and y (this includes the case $z = z'$). Let this common neighbor be w.l.o.g. vertex z. Then, there is a cycle $x - p_1 - \cdots - p_\ell - y - z - x$ in G. Moreover, p_k is not a neighbor of z, vertices x and y are nonadjacent (else they would be connected in $G[M_i]$), and due to π being the shortest path, there are no edges between $x - p_j$ for $j > 1$, $y - p_{j'}$ for $j' < \ell$ and $p_j - p_{j'}$ for $|j - j'| > 1$. Hence, the only chords the cycle might have could be $p_j - z$ edges, with $j \neq k$. But there could only be $l - 1$ such chords and for a cycle of length $l + 3$, l chords are needed to make it chordal. A contradiction.

The remaining case is that z and z' are both not a common neighbor of x and y. We show that there can be no AMO produced by the MCS, when it chooses x or y next from M_i, which would be a contradiction. W.l.o.g., we show the argument for x. Let α be a topological ordering inducing an AMO of G having prefix $\tau_i + (x)$. Denote by $\alpha^{-1}(v)$ the position of v in α. It has to hold that $\alpha^{-1}(x) < \alpha^{-1}(p_1) < \alpha^{-1}(p_2) < \cdots < \alpha^{-1}(p_\ell) < \alpha^{-1}(y)$ as the first vertex p_j with $\alpha^{-1}(p_j) > \alpha^{-1}(p_{j+1})$, would induce a v-structure. As we have $\alpha^{-1}(z') < \alpha^{-1}(p_\ell) < \alpha^{-1}(y)$, vertices z' and p_ℓ have to be adjacent to avoid a v-structure. Then, the same applies to $\alpha^{-1}(z') < \alpha^{-1}(p_{\ell-1}) < \alpha^{-1}(p_\ell)$, which implies an edge between z' and $p_{\ell-1}$. This iteration can be continued only until vertex p_i, which is nonadjacent to z' by assumption. □

Algorithm MCS-ENUM utilizes these results to enumerate the AMOs of a chordal graph. The recursive function `enumerate` first proceeds as the standard MCS by choosing any vertex v from M_i, appending it to τ_i and, in this case recursively, calling `enumerate` to continue in this manner until the AMO induced by τ_n is output. When the recursive call in line 12 returns *for the first time*, however, lines 13-15 are executed computing the set

Algorithm 4.3: Linear-time delay enumeration algorithm MCS-ENUM for listing all AMOs of a chordal graph.

input : Chordal graph $G = (V, E)$.
output: All AMOs of G.

```
1  function enumerate(G, τ_i)
2      if i = n then
3          Output G[τ_n].
4      end
5      // As in Algorithm 2.2, P_i(v) and M_i can be maintained
          efficiently using appropriate data structures
6      foreach v ∈ V do P_i(v) := Ne_G(v) ∩ τ_i
7      M_i := {v ∈ V \ τ_i | for all w ∈ V \ τ_i it holds that |P_i(v)| ≥ |P_i(w)|}
8      v := any vertex from M_i
9      x := v
10     do
11         τ_{i+1} := τ_i + (x)                          // append x to τ_i
12         enumerate(G, τ_{i+1})
13         if x = v then
14             R := {a | a reachable from v in G[M_i]}
15         end
16     while R is non-empty, x := pop(R)
17 end
18 τ_0 := ( )
19 enumerate(G, τ_0)
```

R of vertices reachable from v in $G[M_i]$. These vertices in R are then also chosen as next vertex to visit and for each of them `enumerate` is called recursively as before (whereas set R is only computed once).

Theorem 4.10. *Given a chordal graph G,* MCS-ENUM *enumerates all AMOs of G.*

Proof. Every DAG output in line 3 is an AMO, as it is generated by a linear ordering produced by an MCS. This holds as any chosen vertex is from M_i. To see that the algorithm outputs *all* AMOs of G, recall that every AMO can be represented by an MCS ordering by Lemma 4.7. In principle, Algorithm 4.3 considers all possible courses an MCS could take, except the pruning of vertices unreachable from v. By Lemma 4.9, it suffices to inspect only connected vertices in $G[M_i]$ and by item (2.) of Lemma 4.8 those unreachable vertices would not lead to any new AMO.

Finally, we argue that no AMO is output twice. Every output is obtained by constructing a directed graph based on the ordering given by the graph traversal. Assume for the sake of contradiction that we have two such sequences τ_1 and τ_2 representing the same AMO. Let x and y be the vertices in τ_1 and τ_2 at the first differing position, respectively. Note that x and y are connected and, hence, by item (1.) of Lemma 4.8 it follows that τ_1 and τ_2 yield different AMOs. □

Theorem 4.11. MCS-ENUM *has worst-case delay $O(n + m)$.*

Proof. In a standard MCS, data structures are used, which are able to update M_i in time $O(\Delta(x))$ after visiting vertex x. In our algorithm, we need to efficiently perform the

reverse step, when going back up the recursion tree. It is easy to see that this is also possible in time $O(\Delta(x))$.

Let us partition the steps between two outputs in three phases: (i) the recursion goes "upwards" from an output; (ii) it reaches its "top" in the recursion tree; and (iii) the recursion goes "downwards" towards the next output.

We show that each phase runs in time $O(n + m)$. In phase (i), the do-while loop stops (otherwise we would be in phase (ii) because line 12 would be called, again going "downwards" the recursion tree). Hence, the reachability search (if called) would not yield any vertices and terminate after $O(\Delta(x))$ steps by Lemma 4.9. Overall, the time complexity of a single upwards step is thus $O(\Delta(x))$ and this phase may take time $O(m)$ in total. Phase (iii) is the standard "forward" step of MCS, thus needing time $O(\Delta(x))$, which again amounts to $O(m)$ total time. In phase (ii), non-empty set R is computed, which may take time $O(m)$. However, this phase only consists of a single step. Hence, we obtain a worst-case delay of $O(n + m)$. □

From a practical point-of-view, it is clear that it happens very rarely that the procedure "backs up" far in the recursion tree, most times it will stay close to the leaves between two outputs. Therefore, outputting the resulting AMO is the actual bottleneck of the algorithm and it is faster when only τ is returned in line 3. Formalizing this and analyzing the amortized or average case delay between two outputs is an interesting open problem.

Algorithm 4.4: Linear-time delay enumeration algorithm of Markov equivalent DAGs based on MCS-ENUM.

input : A CPDAG $G = (V, E, A)$.
output: $[G]$.

1 $U(G) \coloneqq (V, E)$, i.e., the graph with only the undirected edges of G
2 Execute Algorithm 4.3 on $U(G)$ and each time add arcs A when outputting the DAG (in line 3 of Algorithm 4.3).

The result for chordal graphs immediately generalizes to CPDAGs as prescribed by Proposition 2.15.

Theorem 1.3. *Let G be a CPDAG. There exists an algorithm which enumerates the members of the MEC $[G]$ with worst-case delay $O(n + m)$.*

Proof. Algorithm 4.3 is called for the graph obtained by removing all directed edges of G. After computing an AMO of this graph, the directed edges can be re-added and the output is a member of $[G]$ produced with delay $O(n + m)$. The correctness follows from Proposition 2.15. Algorithm 4.4 describes this approach. □

4.3 Enumerating Consistent Extensions of PDAGs

In the previous section, we restricted ourselves to the enumeration of consistent extension of CPDAGs. In this section, we consider the problem for general PDAGs.

The set of consistent extensions represented by PDAG G can also be represented by an MPDAG G' which is obtained from G by iteratively applying the four Meek rules (Figure 3.1). In Chapter 3, we showed how this step can be performed in time $O(n^3)$ (or $O(dm)$, where d is the degeneracy of the skeleton, to be more precise). It reduces the enumeration task from PDAGs to MPDAGs, with an additional $O(n^3)$ initialization step.

We also showed in Chapter 3 how a consistent extension of an extendable MPDAG[2] can be computed in time $O(n+m)$. The key insight was to modify the MCS such that, at step i, a vertex from M_i is chosen, which has no incoming edges in arc set A from unvisited vertices, i.e., ones in $V \setminus \tau_i$. We denoted the subset of vertices of M_i satisfying this constraint with M_i^A. This MCS is performed on the *bucket graph* of G. By Lemma 4.7, *every* consistent extension of bucket graph $B(G)$ can be induced by an MCS order (note that the skeleton of $B(G)$ is chordal by Lemma 3.25). Moreover, every consistent extension of G is made up of a consistent extension of $B(G)$ (and the remaining directed edges in G). This constitutes the reverse direction of Lemma 3.23.

Lemma 4.12. *Let $G = (V, E_G, A_G)$ be an MPDAG. Let $D = (V, \emptyset, A_D)$ be a consistent extension of G. Then, graph $C = (V, A_C)$ with $A_C = \{(a,b) \in A_D \mid a - \cdots - b \in G\}$ is a consistent extension of $B(G)$.*

Proof. Assume C is not a consistent extension of $B(G)$. Then, it does not satisfy one of the three conditions of a consistent extension. However, (1.) it is an orientation of $B(G)$ because D is an orientation of G and C is the subgraph on the same skeleton as $B(G)$. Moreover, (2.) it is acyclic as D is acyclic and C is a subgraph. Finally, (3.) it has the same v-structures because $a \to b \leftarrow c$ with a and c nonadjacent in C, but not $B(G)$, would imply the same for graphs D and G (note that a and c cannot be adjacent in G if they are not in $B(G)$). Conversely, a v-structure in $B(G)$ is also in G by (1.). □

On the basis of this, an analogue modification of Algorithm 4.3 suggests itself to enumerate *all* AMOs of a bucket graph $B(G)$: Perform the algorithm on the skeleton of $B(G)$ and only consider vertices in M_i^A, with A being the set of arcs in $B(G)$.

Lemma 4.13. *Let $B = (V, E, A)$ be the bucket graph of an extendable MPDAG and τ_i be a sequence of visited vertices with $i < n$ produced by the modified MCS (function* `bucket-mcs` *in Algorithm 3.1) on* skel(B)*. Then it holds that:*

1. *If $x, y \in M_i^A$ are connected in* skel$(B[V \setminus \tau_i])$*, the set of AMOs produced by choosing x next is disjoint from the set produced by choosing y next.*
2. *If $x, y \in M_i^A$ are unconnected in* skel$(B[V \setminus \tau_i])$*, any AMO produced by choosing y as the next vertex can also be produced by choosing a vertex in M_i^A connected to x in* skel$(B[V \setminus \tau_i])$ *next.*

Proof. For item (1.) consider the shortest path between x and y and assume that it contains directed edges (if not the argument of Lemma 4.8 applies). Then x or y has an incoming arc due to the non-applicability of the first Meek rule in B. This violates the assumption that $x, y \in M_i^A$. For item (2.) the same argument as in Lemma 4.8 holds. □

Reachability can again be tested in a simplified way:

Lemma 4.14. *Let $B = (V, E, A)$ be the bucket graph of an extendable MPDAG and τ_i a sequence of visited vertices with $i < n$ produced by the modified MCS (function* `bucket-mcs` *in Algorithm 3.1) on* skel(B)*. Vertices $x, y \in M_i^A$ are connected in* skel$(B[V \setminus \tau_i])$ *iff they are connected in* skel$(B[M_i^A])$*.*

[2]Throughout this section, we assume that the MPDAGs have a non-empty set of consistent extensions. If needed, this can easily be checked by an $O(n^3)$ preprocessing step.

Proof. Recall that M_i is the set of *all* vertices with maximum number of visited neighbors at step i including the ones with unvisited parents. As shown in the proof of Lemma 4.9 and due to the chordality of skel(B) (Lemma 3.25), the vertices x and y are connected in the subgraph of skel(B) induced by M_i if they are connected in the subgraph induced by $V \setminus \tau_i$. Hence, for the sake of contradiction, assume that x and y are connected in M_i but not in M_i^A. Then there is a shortest path $x = p_1 - p_2 - \cdots - p_{k-1} - p_k = y$ with some $p_j \in M_i \setminus M_i^A$. Consider the p_j with smallest j and assume the shortest path is chosen such that this j is maximized (in case that there are multiple shortest paths). This path is completely undirected in B as x or y would otherwise have an incoming edge by the same argument as in the proof of Lemma 4.13.

Observe that p_j has an incoming edge from an unvisited vertex z_1 in B. By the properties of the MCS we have $z_1 \in M_i$ as otherwise, because $p_j \in M_i$, z_1 would be non-adjacent to a previous neighbor a of p_j implying the v-structure $a \rightarrow p_j \leftarrow z_1$ and violating the fact that the modified MCS returns an AMO. Because B is a bucket graph of an MPDAG, z_1 needs to be adjacent to p_{j-1}. Furthermore, if z_1 has an incoming edge from an unvisited vertex z_2 then this vertex has to be connected to p_{j-1} as well.

This process can be iterated further and we will consider the first z_ℓ that has no unvisited parent. Such a vertex has to exist as otherwise there would be a directed cycle. The edge between z_ℓ and p_{j-1} is undirected by the minimality of p_j (i.e., no vertex before p_j has an incoming edge from an unvisited vertex).

Moreover, z_1 needs to be adjacent to p_{j+1} by the same argument. If this edge is undirected, z_2 needs to be adjacent to p_{j+1}. If it is directed $z_1 \rightarrow p_{j+1}$, then z_1 needs to be adjacent to p_{j+2}. Generally, the highest index vertex on the path that z_s is adjacent to is connected to it by an undirected edge. Meaning z_{s+1} has to be connected to it as well. Thus, z_ℓ is connected by an undirected edge to some p_t on the path, for $t > j$. This yields the desired contradiction as either the path $x = p_1 - \cdots - p_{j-1} - z_\ell - p_t - \cdots - p_k = y$ is shorter than the previously considered one or the first vertex with an unvisited parent has a higher index. □

Using these results, we can show that:

Theorem 4.15. *Let G be an MPDAG. There is an algorithm that enumerates all consistent extension of G with worst-case delay $O(n + m)$.*

Proof. This approach is stated in Algorithm 4.5. We first show that every DAG that is output by the algorithm is a consistent extension of G. This follows from the fact that the algorithms produces outputs just as Algorithm 3.1 (see Theorem 3.27), only with additional backtracking steps.

Moreover, every consistent extension of the given bucket is output. By Lemma 4.12, every consistent extension of G can be obtained by finding a consistent extension of $B(G)$ and by generalization of Lemma 4.7 every such consistent extension can be induced by an MCS ordering. The algorithm considers all MCS orderings, except (i) those which do not conform with the given directed edges (if vertex v has an incoming arc from an unvisited vertex w it cannot be in M_i^A) and (ii) those vertices that are unreachable from v (which do not produce new consistent extensions by item (2.) of Lemma 4.13).

Finally, it remains to show that no consistent extension is output twice. Clearly, every sequence τ that we obtain is different. So assume for the sake of contradiction that the algorithm outputs two sequences τ and τ' that represent the same consistent extension. Let x and y be the vertices in τ and τ' at the first differing position. Then x and y are connected in the induced subgraph over the unvisited vertices (by line 15) in the skeleton of B. A contradiction since this implies that the AMOs represented by τ and τ' differ.

For the complexity analysis we partition the algorithm into three phases (i), (ii), and (iii) as in the proof of Theorem 4.11. The cost of (i), (ii) and (iii) are the same as in the analysis of Theorem 4.11. Particular, observe that, similar as before, reachability terminates after $O(\Delta(x))$ steps for vertex x in phase (i) due to Lemma 4.14. □

Algorithm 4.5: Linear-time delay enumeration algorithm of the consistent extensions of an MPDAG.

input : MPDAG $G = (V, E, A)$.
output: All consistent extensions of G.

function `bucket-enumerate`(B, A, τ_i, A_G)
  if $i = n$ **then**
    Output graph obtained by orienting B according to τ_n and adding the arcs from A_G (which are not present already).
  end
  `// As in Algorithm 3.1, the next three lines can be implemented efficiently with appropriate data structures`
  foreach $v \in V$ **do** $P_i(v) \coloneqq Ne_B(v) \cap \tau_i$
  $M_i \coloneqq \{v \in V \setminus \tau_i \mid \text{for all } w \in V \setminus \tau_i \text{ it holds that } |P_i(v)| \geq |P_i(w)|\}$
  $M_i^A \coloneqq \{v \in M_i \mid \text{there exists no } w \in V \setminus \tau_i \text{ such that } w \to v\}$
  $v \coloneqq$ any vertex $x \in M_i^A$
  $x \coloneqq v$
  do
    $\tau_{i+1} \coloneqq \tau_i + (x)$ `// append` x `to` τ_i
    `bucket-enumerate`(B, τ_{i+1})
    if $x = v$ **then**
      $R \coloneqq \{a \mid a \text{ reachable from } v \text{ in } B[M_i^A]\}$
    end
  while R *is non-empty*, $x \coloneqq pop(R)$
end
$B(G) \coloneqq (V, E_G, A_{B(G)})$ with $A_{B(G)} = \{(a, b) \in A_G \mid a - \cdots - b \in G\}$
$\tau_0 \coloneqq (\,)$
`bucket-enumerate`$(\text{skel}(B(G)), A_{B(G)}, \tau_0, A_G)$

Theorem 4.16. *There is an algorithm that enumerates all consistent extensions of a given PDAG with $O(n + m)$ delay after an initialization step of time $O(n^3)$.*

Proof. The graph is initially transformed into its MPDAG. This is possible in time $O(n^3)$ as shown in Theorem 3.30. Afterwards, apply Algorithm 4.5. □

4.4 Enumerating Markov Equivalent DAGs With Small Changes

The results of the previous sections settle the worst-case complexity of enumerating the members of an MEC.[3] In this section, we complement these results with an enumeration

[3] At least if every DAG is output separately, a run-time of $o(n + m)$ is not achievable as this would be less than the size of the graph. There are however, algorithms which modify the configurations in place and this allows for faster, even constant-time delay, enumeration algorithms.

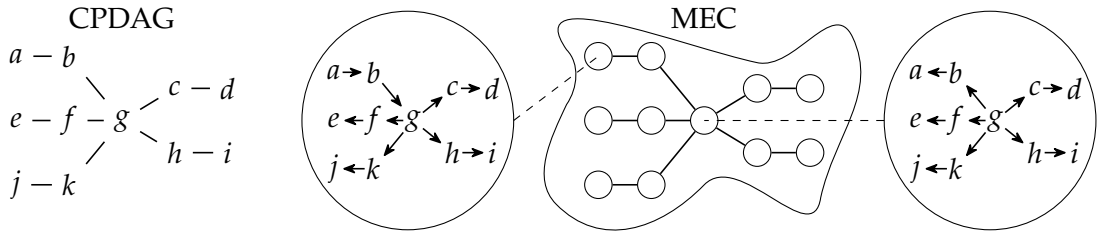


Figure 4.3: An example CPDAG G for which there exists no enumeration sequence of $[G]$ with successive DAGs having SHD smaller or equal to two. Two DAGs in the MEC are connected by an edge if they can be transformed into each other by a single edge reversal. For trees, the resulting topology coincides with the one of the CPDAG, each DAG in the MEC can be represented by its unique source vertex. During the enumeration, the DAG in the center can be used only once, which makes it impossible to list all "leaf DAGs" when allowing only for distance at most two.

sequence of small changes between consecutive DAGs. While this approach has worse delay, such sequences are more natural from the causal perspective, with only a few edge orientations changing at a time, and emphasizes the structural properties of Markov equivalence. In more detail, we show that all graphs in an MEC can be enumerated in a sequence such that every two consecutive DAGs have structural Hamming distance (SHD) at most three. Our results are based on the seminal characterization of Markov equivalence stated in Theorem 4.4 above, which is the basis for CHICKERING-ENUM.

It implies that is possible to go from one member of an MEC to another with only single edge reversals and this also holds for consistent extensions of PDAGs and MPDAGs. The task we are trying to solve, however, is to *enumerate all* Markov equivalent DAGs, meaning the goal is to find a sequence in which *every* DAG occurs exactly once. It can be shown that such a sequence with SHD at most one does indeed *not* exist. Figure 4.3 provides an example that does not even allow a sequence of SHD *two.*

Lemma 4.17. *There exists an MEC $\mathcal{M}$ without a sequence $(D_1, D_2, \dots)$, which contains each DAG in $\mathcal{M}$ once and has the property that successive graphs have an SHD of at most two.*

However, if we permit *three* edge reversals between consecutive DAGs we can always find such a sequence. This relies on the result by [Sekanina, 1960], that any *connected* graph has a sequence containing each node exactly once such that successive nodes in the sequence have distance ≤ 3 in the graph. The graph we consider in this context is the one, which contains the DAGs in an MEC as nodes[4] and an edge between nodes D_1 and D_2, if the corresponding DAGs $\mathrm{SHD}(D_1, D_2) = 1$. By Chickering [1995], this graph is connected, which implies Corollary 4.18 below. Utilizing the result by Sekanina [1960] for enumeration in this way is generally possible by constructing the graph with all configurations as nodes and connecting them if they have distance one.[5] Consequently, it is an essential tool for constructing enumeration sequences with small distance (also known as Gray codes) in general, for a reference see Section 2.4 of [Mütze, 2022].

[4]We use the term node instead of vertex here to avoid confusion with the vertices of the DAGs.

[5]Depending on the context, different measures of distance might be useful. E.g., when enumerating permutations, two permutations may have distance one if they can be transformed into each other by a single swap of objects.

Corollary 4.18. *Every MEC can be represented as sequence $(D_1, D_2, \dots)$ of Markov equivalent DAGs such that successive graphs have SHD at most three.*

Proof. For a constructive, proof consider the graph that contains all DAGs in the MEC as nodes. In that graph connect two nodes with an edge if the DAGs can be transformed into each other by a single edge reversal (hence, these have SHD one). By Chickering [1995] (Theorem 4.4 above), the graph is connected.

Every connected graph has a sequence $(p_1, p_2, \dots)$ that contains every node exactly once such that the distance between consecutive nodes is at most three [Sekanina, 1960, Mütze, 2022]. This sequence can be constructed by performing a depth-first-search (DFS) starting at an arbitrary node r and appending nodes with an even distance from r in the DFS tree when they are discovered and nodes with an odd distance from r when they are fully processed (essentially mixing pre- and post-order depending on the layer of the DFS tree). The SHD between two output nodes is never larger than three: When going down the DFS tree, every second node is output, when going up (after last outputting in odd layer i) the node in layer $i-2$ is output after it is finished. Hence, if it has no unvisited neighbors, the SHD is two. If it does, one of these gets explored and, hence, immediately output as it is in even layer $i-1$. In this case, the SHD to the last output is three. □

The approach used in the proof is formalized in Algorithm 4.6.

Algorithm 4.6: Enumeration algorithm with SHD $\leq$ 3 of Markov equivalent DAGs based on Chickering's characterization [Chickering, 1995] (SHD3-ENUM).

input : A CPDAG $G = (V, E, A)$.
output: $[G]$.

```
1  D := any DAG in [G]
2  vis := {D}
3  enumerate(D, vis, 0)
4  function enumerate(D, vis, i)
5      if i mod 2 = 0 then
6          Output D
7      end
8      foreach DAG D' obtained by reversing a covered edge in D do
9          if D' ∉ vis then
10             vis := vis ∪ {D'}
11             enumerate(D', vis, i + 1)
12         end
13     end
14     if i mod 2 = 1 then
15         Output D
16     end
17 end
```

Theorem 4.19. *For a CPDAG G,* SHD3-ENUM *enumerates the DAGs in $[G]$ with delay $O(m^2)$ such that successively output DAGs have SHD at most three.*

Proof. The argument is similar to the proof of Theorem 4.5. First, it is clear that all DAGs in the MEC are output exactly once. Moreover, by the proof of Corollary 4.18, it satisfies the stated property with regard to the SHD.

The analysis of the delay differs slightly in the sense that this algorithm has worst-case delay $O(m^2)$ (instead of $O(m^3)$) due to the fact that between two outputs only a constant number of recursive calls are handled. This follows from the fact that the algorithm outputs successive DAGs with SHD ≤ 3 and that between the output of these DAGs, there are only constantly many steps in the traversal of the MEC. The cost per DAG can be bounded by $O(m^2)$ as in Theorem 4.5. □

Both Algorithm 4.6 as well as Theorem 4.19 directly apply to PDAGs as well. In contrast to these results, we note that MEEK-ENUM, CHICKERING-ENUM and MCS-ENUM do not enumerate an MEC such that the SHD between two successive outputs is at most three.

Lemma 4.20. *Sequences of DAGs produced by* MEEK-ENUM, CHICKERING-ENUM *and* MCS-ENUM *may contain consecutive DAGs with SHD larger than three.*

Proof. Consider the CPDAG shown in Figure 4.3. Since MEEK-ENUM has no preferences on the edge it orients first, it may start with the edge $a \rightarrow b$. All other edge directions would then follow from the first Meek rule yielding the output DAG shown in Figure 4.3. The orientation $a \leftarrow b$ is tried afterwards, which would result in no further directed edges. Then, assume the next undirected edge picked by the algorithm is the ones between h and i. It may be oriented as $h \leftarrow i$ yielding a DAG with SHD four to the previously output DAG.

Similar arguments hold for CHICKERING-ENUM and MCS-ENUM, the former could end up in a state where the only DAGs left are the one with edge $a \rightarrow b$ and the one with $h \leftarrow i$ and the latter could start with vertex a and afterwards choose i as the first vertex – yielding again the same DAGs. □

Computationally, our results do not imply a better bound (compared to e.g. the linear-time algorithm we presented above) on the delay in producing the sequence from Corollary 4.18 and we leave this as an open problem. The constructive algorithm (which we call SHD3-ENUM) given in the proof of Corollary 4.18 behaves relatively similar to CHICKERING-ENUM and has delay $O(m^2)$ as every DAG may have m neighbors and we have to check for each of them whether they were already visited. It seems unlikely that this can be improved without further structural insights.

Lastly, we remark that the same idea can also be used in the more general setting of enumerating Markov equivalent maximal ancestral graphs (MAGs) without selection bias, which are causal models allowing for latent confounders and for which a similar transformational characterization exists [Zhang and Spirtes, 2005]. A detailed treatment of MAGs is given in Chapter 7. We focus here on the following property generalizing the results by Chickering [1995].

Theorem 4.21 (Zhang and Spirtes [2005])**.** *Two MAGs without selection bias G and G' are Markov equivalent iff there exists a sequence of single edge mark changes in G such that*

1. *after each mark change, the resulting graph is a DMAG and is Markov equivalent to G,*
2. *after all the mark changes, the resulting graph is G'.*

It follows that the graph over Markov equivalent DMAGs with an edge between DMAGs with SHD one (we define the SHD similar to the DAG case, that is, the number of differing edges; one could also define it to be the number of differing edge marks, and the statement still holds) is connected and hence, there is, by the same argument as in Corollary 4.18, a sequence of DMAGs containing each in the MEC exactly once with successive DMAGs having SHD ≤ 3.

Corollary 4.22. *Every MEC of MAGs without selection bias can be represented as sequence $(M_1, M_2, \dots)$ of Markov equivalent MAGs with SHD between successive graphs at most three.*

In contrast, approaches such as MEEK-ENUM do not exist for MAGs as analogue rules to the ones by Meek [1995] proposed by Zhang [2008b] only complete the graph in case the edge marks are inferred from observational data. If edge marks are chosen for the sake of enumeration, these rules are not known to be complete. Generally, it is an interesting direction for future work to investigate the computational aspects of the enumeration of MECs of MAGs.

Finally, we note that the transformational characterization in Theorem 4.21 does not hold for MAGs *with* selection bias and it follows from the example given in Zhang and Spirtes [2005] that it is indeed not possible to find an enumeration sequence with SHD ≤ 3 in this case.

4.5 Experimental Evaluation

In addition to the theoretical results, we also show that MCS-ENUM and its generalizations are practically implementable and significantly faster than previously used algorithms.[6] In Figure 4.4, we compare the average delay of the four approaches (MEEK-ENUM, CHICKERING-ENUM (abbreviated CHI. in the figure), MCS-ENUM, SHD3-ENUM) implemented[7] in Julia [Bezanson et al., 2017]. For each instance (the generation procedure is described below), the programs were terminated after two minutes if the enumeration was not completed. For the experiments, we used a single core of the AMD Ryzen Threadripper 3970X 32-core processor on a 256 GB RAM machine. As the enumeration problem reduces to listing the AMOs of a chordal graph (as implied by Proposition 2.15 and Corollary 2.19), we consider as instances undirected graphs generated by randomly inserting edges, which do not violate chordality, until a graph with $k \cdot n$ edges is reached. Note that these instances are all CPDAGs, just fully undirected ones, and thus all approaches can be applied to this setting.[8]

The results clearly show that MCS-ENUM is by far the fastest among the algorithms. This is mainly due to the fact that the other algorithms always incur a cost of at least $\Omega(n + m)$, whenever a single edge is (re-)oriented. MEEK-ENUM needs to apply the completion rules, whereas CHICKERING-ENUM and SHD3-ENUM require checking whether the resulting DAG was already output (which might often be the case). Still, the latter algorithms are significantly faster than MEEK-ENUM (at the cost of higher memory demand), and notably have both very similar delay, further showing that the enumeration

[6] The implementations of the algorithms are available at `https://github.com/mwien/mec-enum`.

[7] The algorithm MEEK-ENUM is implemented as in, e.g., the package `TETRAD`, with a direct application of the Meek rules and not using the improved algorithm used in Theorem 3.30. The overall results would hardly be affected, even when using the $O(n^3)$ algorithm instead.

[8] Wienöbst et al. [2023] also compare the results for CPDAGs *with* directed edges as well as for PDAGs, which both lead to very similar results. Generally, MCS-ENUM is constrained mostly by constructing the output graph from the MCS ordering. The other approaches are more global and much slower with the concrete run-time depending mostly on graph size and density.

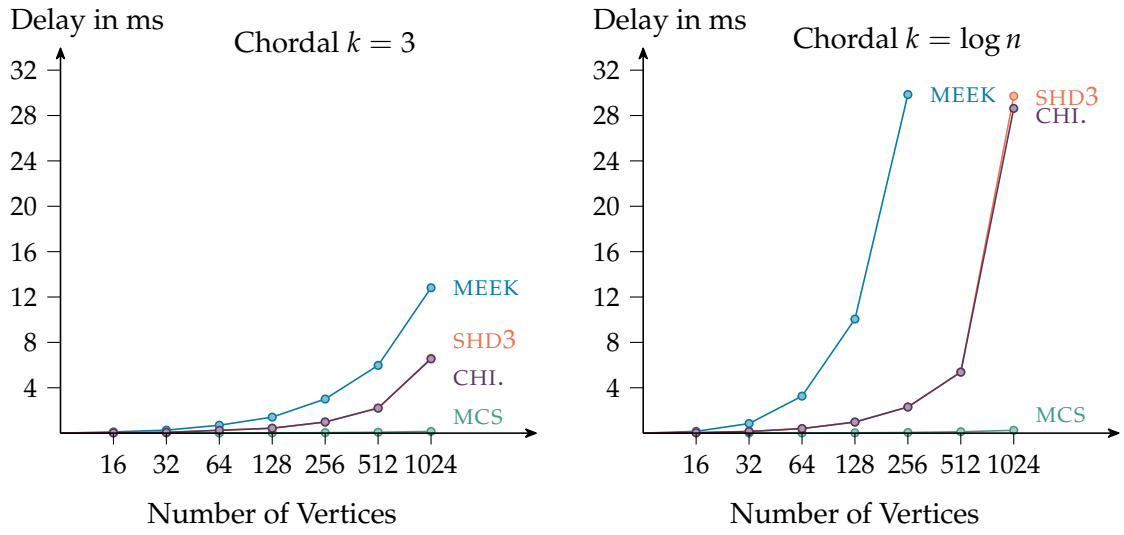


Figure 4.4: Average delay in milliseconds for enumerating the AMOs of random chordal graphs with $m = k \cdot n$ edges. The graphs are generated by starting with a tree over n vertices sampled uniformly and adding edges between randomly chosen pairs of non-adjacent vertices until the graph has $k \cdot n$ edges. We compare the algorithms MEEK-ENUM, CHICKERING-ENUM (CHI.), MCS-ENUM and SHD3-ENUM.

with SHD at most three gives mainly structural insights into Markov equivalence and has in itself no computational advantage.

4.6 Conclusions

We have given the first formal and exhaustive treatment of the fundamental problem of enumerating Markov equivalent DAGs. Our main results are twofold: (i) we significantly improve the run-time of enumeration by giving the first linear-time delay algorithm, which is also practically effective, and (ii) we give structural insights into Markov equivalence by constructing an enumeration sequence with minimal distance between successive graphs. The concepts for (ii) are general and directly apply to MAGs without selection bias as well.

As an open problem, it remains to find more efficient enumeration algorithms for MAGs, where, currently, approaches in the spirit of both MEEK-ENUM and MCS-ENUM are not known.

5

Counting Markov Equivalent DAGs

In the previous section, we discussed the computational aspects of enumerating all consistent extensions of a CPDAG G, i.e., the task of enumerating all DAGs in the corresponding MEC. A closely related problem is to compute the size of a Markov equivalence class. Indeed, one could solve it with brute-force by enumerating of all its members using Algorithm 4.3 and counting them one-by-one. However this would not lead to a polynomial-time algorithm with $|[G]|$ being worst-case exponentially large in the number of vertices, taking size $O(n!)$ if G is the complete graph on n vertices. In this chapter, we propose a more intricate algorithmic approach, which constitutes the first polynomial-time algorithm for computing the size of an MEC.

The size of an MEC is one of its most fundamental properties and naturally of high interest in causal discovery as it quantifies uncertainty about the true underlying causal structure. There are multiple practical applications of an efficient counting algorithm, for example, to sample a DAG uniformly from the MEC, in active learning of DAGs through interventions [He and Geng, 2008, Hauser and Bühlmann, 2014] and in causal effect estimation over MECs [Maathuis et al., 2009]. We devote Chapter 6 to these applications and focus solely on the algorithmic aspects of the counting task in this chapter.

The problem has a long history and it was first mentioned in Section 4.1 of the seminal work by Meek [1995] on causal explanations with background knowledge. Therein, it is mentioned that a method for calculating the size of a Markov equivalence class may be derived from the proofs of the main theorems of the paper. Indeed, the proofs contain powerful results and already show that the problem reduces to finding consistent extensions of the chordal components of G (as we will discuss in detail below). They also highlight the importance of the clique-tree representation of a chordal graph, which we build on extensively. However, from these properties only an exponential-time algorithm follows, which amounts to brute-force enumeration of the DAGs for any connected component of the undirected subgraph in order to compute its size. The number of DAGs in the MEC is then the product of this quantity for each chordal component. The algorithm was later described and analyzed in Autio et al. [1999].

This approach can be effective in case the CPDAG has few undirected edges or its undirected subgraph has small connected components. One example is when each such component consists of a single undirected edge. Then, even though there would be 2^k (with k being the number of components) DAGs in the Markov equivalence class, the algorithm would be be extremely fast (it is easy to enumerate the two orientations of each component and to compute the product). For large undirected components, however, this method exhibits a worst-case exponential time complexity.

Starting with the pivotal paper by He et al. [2015], the problem regained significant interest from the research community. In particular, the worst-case time complexity became the main topic of interest. With better and better, but still worst-case exponential-time, counting algorithms [Talvitie and Koivisto, 2019, Ghassami et al., 2019, Ganian et al., 2020, AhmadiTeshnizi et al., 2020] the question emerged whether this problem might be solvable in polynomial time. In this work, we answer this question affirmatively by presenting an algorithm with worst-case run-time $O(n^4)$. This algorithm, which we name *Clique-Picking*, certifies that the problem can be solved efficiently in theory and practice, even for its hardest instances.

5.1 Background

Main Problem. We consider the following computational problem (recall that $\text{EXT}(G)$ is the set of consistent extensions of graph G):

Problem 5.1. *#EXT*

Instance: *Graph $G = (V, E, A)$.*
Result: *Number of consistent extensions of G, that is $|EXT(G)|$.*

In this chapter, we focus on the case that G is a CPDAG, in which the problem amounts to computing the size of an MEC (in contrast to computing the size of a subclass of an MEC for general graphs G).

From Proposition 2.15, it follows that only the undirected edges of G need to be considered for computing $|\text{EXT}(G)|$ and that they form a chordal graph $U(G)$. We have not relied on this fact before, but it is obvious that the connected components of $U(G)$ (we denote those as $\mathcal{C}(G)$ instead of the more verbose $\mathcal{C}(U(G))$) can be considered separately. This observation is crucial for obtaining efficient counting algorithms. Consequently, we define $|\text{AMO}(H)|$ as the number of AMOs of chordal graph H and throughout this chapter assume that this graph is *connected*. This is formalized in the following proposition:

Proposition 5.2 (Gillispie and Perlman [2002], He and Geng [2008])**.** *Let G be a CPDAG. Then,*

$$|EXT(G)| = \prod_{H \in \mathcal{C}(G)} |AMO(H)|. \tag{5.1}$$

Thus, the problem #EXT of counting the number of DAGs in an MEC reduces to counting the number of AMOs in a connected chordal graph [Gillispie and Perlman, 2002, He and Geng, 2008] as, given a polynomial-time algorithm for the latter task, Equation 5.1 can be evaluated in polynomial-time as well. Therefore, we focus on the following simpler problem in this chapter:

Problem 5.3. *#AMO*

Instance: *Connected chordal graph $G = (V, E)$.*
Result: *Number of acyclic moral orientations of G, that is $|AMO(G)|$.*

In terminology from the chordality literature, this amounts to computing the number of different orientations of a chordal graph, which are induced by a perfect elimination ordering of the vertices. This problem has not been studied in the chordal graph community, only in the modern works on counting Markov equivalent DAGs, even though orientations induced by PEOs are discussed, e.g., by Vandenberghe and Andersen [2015].

Table 5.1: We state the time complexities of fundamental orientation tasks on chordal and general graphs. Only undirected graphs are considered here.

	Graph type		
Orientations	Chordal	General	Remarks
All	$O(m)$	$O(m)$	Answer is 2^m.
Acyclic	$O(n+m)$	#P-hard	Given by $\lvert\chi_G(-1)\rvert$ [Stanley, 1973].
Acyclic moral	$O(n^4)$	$O(n^4)$	Stated in Theorem 5.34 and Theorem 5.43.

Counting orientations of graphs. Counting orientations of undirected graphs is a fundamental combinatorial task. It is interesting to contrast the task of finding the number of acyclic and moral orientations with related ones. An overview is given in Table 5.1. Obviously, the number of all possible orientations is simply 2^m, with m denoting the number of edges of the graph. More interesting is to count only orientations which are acyclic. Counting those can be shown to be #P-hard for general graphs [Linial, 1986]. This result follows from deep connections between acyclic orientations and graph polynomials from algebraic graph theory. The core insight is that the number of acyclic orientations is given by $|\chi_G(-1)|$ [Stanley, 1973], where χ_G is the chromatic polynomial of graph G, i.e., the unique polynomial of degree n for which $\chi_G(k)$ gives, for positive k, the number of k-colorings of G. The more general Tutte polynomial T_G yields the number of acyclic orientations when evaluated at $T_G(2,0)$ [Welsh, 1999]. These connections imply the positive result for the subclass of chordal graphs because here the chromatic polynomial can be computed in linear-time, as it is given by $\chi_G(k) = \prod_i (x - |P_i(v_i)|)$ [West, 2001] for PEO $\tau = (v_1, v_2, \ldots, v_n)$ and $P_i(v_i)$ being the set of neighbors of v_i coming before it in τ in analogy to the notation introduced in Section 2.3, leading to a polynomial-time algorithm for counting the number of acyclic orientations.

One might wonder how the problem of counting acyclic *moral* orientations fits into this landscape. Firstly, by Corollary 2.18, only chordal graphs have AMOs meaning the problem cannot differ in hardness between chordal and general graphs. Moreover, one can easily show that the number of AMOs *cannot* be expressed as a functional of the Tutte polynomial, see Figure 5.1 for a counterexample. This implies that counting AMOs is not possible based on a contraction-deletion recurrence, as the Tutte polynomial is the most general such invariant [Godsil and Royle, 2001]. Hence, we can conclude that different tools are needed for tackling the problem #AMO, which we will derive in the following.

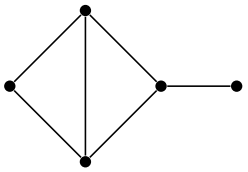
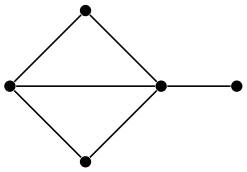
Related work. Before, it is helpful to first review the work by He et al. [2015] as it contains fundamental ideas for efficiently counting AMOs. The goal is to express $|\mathrm{AMO}(G)|$ recursively. For this, a few observations are needed.

Lemma 5.4 (He et al. [2015])**.** *Let D be an AMO of connected chordal graph G. Then, D has exactly one source vertex, i.e., a vertex with $\delta_G^-(v) = 0$.*

Denote the set of AMOs of G, which have s as source with $\mathrm{AMO}(G, s)$.

Definition 5.5 (Source orientation [He et al., 2015])**.** *Let G be a connected chordal graph and $s \in V$. We define $G^s := \mathrm{pdag}(AMO(G,s))[V \setminus \{s\}]$.*

Hence, G^s is the graph over vertices $V \setminus \{s\}$, which contains edge $a - b$ if there are $D_1 \in \mathrm{AMO}(G,s)$ with $a \to b$ and $D_2 \in \mathrm{AMO}(G,s)$ with $a \leftarrow b$, and which contains $a \to b$ if there is $D_1 \in \mathrm{AMO}(G,s)$ with $a \to b$ but no D_2 with $a \leftarrow b$ (see Definition 2.6). It is

12 AMOs   13 AMOs

$$T(x,y) = x^2 + 2x^3 + x^4 + xy + 2x^2y + xy^2$$
$$\chi(x) = x^5 - 6x^4 + 13x^3 - 12x^2 + 4x$$

Figure 5.1: Two chordal graphs with the same Tutte and chromatic polynomial $T(x,y)$ and $\chi(x)$, but with a different number of AMOs.

rather easy to observe that the number of consistent extensions of G^s equals $|\text{AMO}(G,s)|$, that is the number of AMOs of G with s as source vertex. Based on this and in combination with Lemma 5.4, He et al. [2015] derive the following recursive formula:

Proposition 5.6 (He et al. [2015])**.** *Let $G = (V,E)$ be a connected chordal graph. Then,*

$$|AMO(G)| = \sum_{s \in V} \prod_{H \in \mathcal{C}(G^s)} |AMO(H)|$$

This proposition utilizes the fact that the number of AMOs with source vertex s can again be expressed as the number of AMOs of connected chordal graphs, namely $\mathcal{C}(G^s)$, the connected components of (the undirected subgraph of) G^s. This is due to the fact that (i) $\mathcal{C}(G^s)$ are again chordal and (ii) can be oriented independently of each other and the directed edges to yield exactly the consistent extensions of G^s.

Every H in the recursive formula in Proposition 5.6 is an induced subgraph of G. As there are 2^n induced subgraphs, evaluating this expression and storing $|\text{AMO}(\cdot)|$ whenever it is first computed yields a run-time of $O(2^n \cdot \text{poly}(n))$, which improves upon naive $O(n!)$ [Talvitie and Koivisto, 2019]. This worst-case complexity barrier was not broken until the development of the first polynomial-time algorithm presented below.

5.2 The Clique-Picking Algorithm

Every AMO of chordal graph G can be represented by at least one topological ordering τ, which is a PEO (Lemma 2.17). The underlying idea of our counting approach is to associate *exactly* one topological ordering to each AMO of G. For this, it is useful to consider only topological orderings, which are well-behaved in the following sense (in this chapter we often write π_S to denote an arbitrary permutation or linear ordering of set S):

Definition 5.7 (Clique-starting ordering)**.** *Let G be a connected chordal graph. A topological ordering τ is called* clique-starting *if it has a maximal clique as prefix, i.e., $\tau = \pi_K + \pi_{V \setminus K}$ with K being a maximal clique.*

We denote all clique-starting topological orderings representing an AMO D of a graph G by $\text{top}(D) = \{\tau_1, \tau_2, \dots\}$ 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 5.8. *Every AMO can be represented by a clique-starting topological ordering.*

Proof. Consider AMO D. 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 (Lemma 5.4) 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 $u \to v$ in D 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 arcs from unvisited vertices (i.e., by Kahn's algorithm).

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 arc from a vertex in $V \setminus S$. Note that no vertex in $w \in W$ can have an incoming edge from $x \in V \setminus (S \cup W)$ as this would imply a v-structure $y \to w \leftarrow x$ for a $y \in S$ not adjacent to x in the given graph D (vertex y exists because $x \notin W$). As the graph D is acyclic (and this property holds for taking induced subgraphs, i.e., for $G[W]$ as well by Lemma 3.10) there has to be a vertex in W with no incoming arc from a vertex in $V \setminus S$ – a contradiction. □

Based on this observation, we generalize the definition of G^s (Definition 5.5) with the goal of handling whole cliques at once: For this, we consider permutations π_S of a clique S, as each π_S is associated with a distinct AMO of the subgraph induced by S.[1] First, we generalize $\text{AMO}(G, s)$, that is the set of AMOs with source s, to sets S and sequences σ.

Definition 5.9 ($\text{AMO}(G, \sigma)$ and $\text{AMO}(G, S)$)**.** *Let G be a connected chordal graph, S be a set of vertices and σ a linear ordering of S. We define $AMO(G, \sigma) := \{D \in AMO(G) \mid \exists \tau = \sigma + \pi_{V \setminus S}$ with $G[\tau] = D\}$ and analogously $AMO(G, S) := \{D \in AMO(G) \mid \exists \tau = \pi_S + \pi_{V \setminus S}$ with $G[\tau] = D\}$.*

Similarly, we define $\text{top}(D, S)$ to be the topological orderings representing D having some permutation of S as prefix and $\text{top}(D, \sigma)$ the ones, which have precisely σ as prefix. Moreover, in analogy to G^s introduced above, we define G^S and G^σ.

Definition 5.10 (G^σ and G^S)**.** *Let G be a connected chordal graph, S be a clique in G, and σ be a permutation of S. We define*

1. $G^\sigma := \text{pdag}(AMO(G, \sigma))[V \setminus S]$ *and*
2. $G^S := \text{pdag}(AMO(G, S))[V \setminus S]$.

As in case of G^s, the number of consistent extensions of G^σ corresponds to $|\text{AMO}(G, \sigma)|$. We give a short proof.

Lemma 5.11. *Let G be a connected chordal graph, S a clique in G and σ a linear ordering of S. Then, $|EXT(G^\sigma)| = |AMO(G, \sigma)|$.*

Proof. Observe that all DAGs in $\text{AMO}(G, \sigma)$ have the same skeleton and v-structures (namely none), which translates to their PDAG representation $\text{pdag}(\text{AMO}(G, \sigma))$. The set of consistent of this graph yields precisely all DAGs with the same skeleton as G, no v-structures and σ as prefix of every topological ordering, which coincides with $\text{AMO}(G, \sigma)$. □

[1] We try to be consistent in using K for *maximal* cliques and S for cliques in general. This will be useful below, where we consider minimal separators, which are (non-maximal) cliques and to be distinguished from maximal ones.

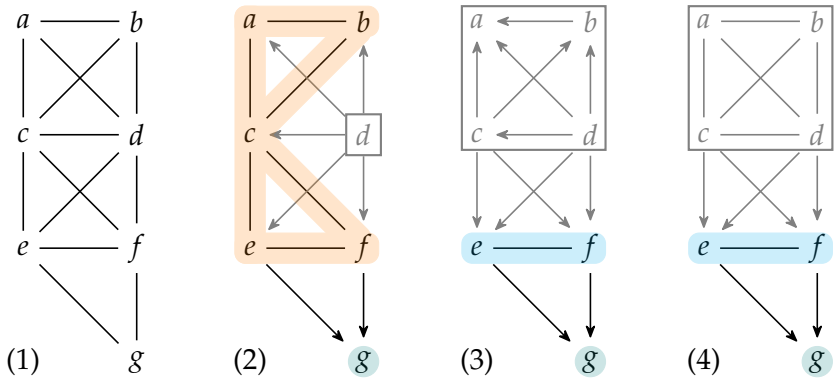


Figure 5.2: For connected chordal graph G in (1), the figure shows $G^{(d)}$ in (2), $G^{(d,c,b,a)}$ in (3), and $G^{\{a,b,c,d\}}$ in (4). $\mathcal{C}(G^d)$, $\mathcal{C}(G^{(d,c,b,a)})$ and $\mathcal{C}(G^{\{a,b,c,d\}})$ are indicated by the colored regions and the vertices put at the beginning of the topological ordering by a rectangle (all edges from the rectangle point outwards). Vertices inside the rectangle and edges incident to them are grayed out, since they are not part of G^s, G^σ and G^S.

Our aim is to develop a recursive formula generalizing Proposition 5.6 towards handling cliques as "source" (instead of single vertices). Towards this, we derive central properties of G^σ and G^S, namely the invariance to the choice of permutation of S. For obtaining an efficient algorithm, the immediate corollary that the connected components $\mathcal{C}(G^\sigma)$ are independent of the permutation σ is crucial.

Lemma 5.12. *Let G be a connected chordal graph, S be a clique in G, and σ and σ' permutations of S.*

1. $G^\sigma = G^{\sigma'}$,
2. $G^\sigma = G^S$ *and*
3. $\mathcal{C}(G^\sigma) = \mathcal{C}(G^S)$.

Proof. We prove (1.), from which the other two statements follow immediately. To this end, consider $D \in \mathrm{AMO}(G, \sigma)$. We will show that there exists $D' \in \mathrm{AMO}(G, \sigma')$ with $D[V \setminus S] = D'[V \setminus S]$. This will immediately imply the statement.

Let τ be any topological ordering of D, which starts with σ, that is $\tau = \sigma + \pi_{V \setminus S}$. Then, consider $\tau' = \sigma' + \pi_{V \setminus S}$ and $D' = G[\tau']$. To conclude the proof we show that D' is an AMO of G. By definition it is an acyclic orientation. Assume for the sake of contradiction that it contains a v-structure $a \to b \leftarrow c$. Because D does not and only directions between vertices in S have been changed in D', it has to hold that either

1. two vertices of a, b, c are in S (w.l.o.g. assume these are a and b), but then we have $b \to c \notin S$ as c is not in S and thus preceded by b in τ', or
2. all three vertices are in S, but then $a \to b \leftarrow c$ is no induced subgraph with S being a clique. □

Figure 5.2 gives example graphs G^s, G^σ and G^S. For G in (1), a clique $S = \{a, b, c, d\}$, and a permutation $\sigma = (d, c, b, a)$, graph G^σ is presented in (3). It is the subgraph induced by vertices $\{e, f, g\}$ of the union of two AMOs of G, whose topological orderings begin with σ. The first DAG can be induced by topological ordering (d, c, b, a, e, f, g) and the

second one by (d, c, b, a, f, e, g). Graph G^σ is identical to G^S, shown in (4), as prescribed by Lemma 5.12. In contrast, G^s for $s = d$, given in (2), leads to significantly more remaining vertices and undirected edges.

As starting point towards developing a polynomial-time algorithm for #AMO, we construct a linear-time algorithm for computing $\mathcal{C}(G^S)$. It is given in Algorithm 5.1 and yields structural insights we will later use for deriving a recursive formula for counting Markov equivalent DAGs.[2]

Algorithm 5.1: Computing $\mathcal{C}(G^S)$.

input : Connected chordal graph $G = (V, E)$, a clique $S \subseteq V$.
output: $\mathcal{C}(G^S)$.

```
1  τ_0 := ( )
2  T_0 := S
3  for i ← 0 to n − 1 do
4      if i < |S| then
5          x := any element of S
6          Remove x from S
7      else
8          // As in Algorithm 2.2, P_i(v) and M_i can be maintained
               efficiently using appropriate data structures
9          foreach v ∈ V do P_i(v) := Ne_G(v) ∩ τ_i
10         M_i := {v ∈ V \ τ_i | for all w ∈ V \ τ_i it holds that |P_i(v)| ≥ |P_i(w)|}
11         x := any element of M_i
12         Append connected components of G[M_i \ T_i] to the output
13         T_{i+1} := T_i ∪ M_i
14     end
15     τ_{i+1} := τ_i + x                                  // append x to τ_i
16 end
```

Let $t(x)$ denote the first i such that x is in T_i but not in T_{i-1}. The ordering τ_n is not needed in the algorithm except for the definition of M_i (which is maintained dynamically in a concrete implementation, see Section 2.3). We include it explicitly to prove the following lemma.

Lemma 5.13. *Let G be a chordal graph and τ_n the visit order computed by Algorithm 5.1. Then, τ_n is a PEO of the vertices of G.*

Proof. By Proposition 2.21, the statement holds if for any chosen x, it is true that $x \in M_i$. This is trivially the case in line 11. It remains to show that it also holds in line 5. Observe that the first $|S|$ chosen vertices are all from clique S. Hence, when a vertex x from S is chosen *all* previously chosen vertices are neighbors of x, implying it is in M_i. □

Theorem 5.14. *Given connected chordal graph G and clique S, Algorithm 5.1 computes $\mathcal{C}(G^S)$. It can be implemented to run in time $\mathcal{O}(n + m)$.*

[2] A naive approach for would be to choose an arbitrary permutation π_S and compute G^{π_S} by setting the initial orientations between edges in S and from S to its neighbors and afterwards applying the Meek rules (Figure 3.1). Indeed, computing G^{π_S} is an instance of the maximal orientation problem studied in Chapter 3. In this section we show that in this special case it is possible to solve the problem in linear-time (also see the subsequent chapter for generalizations of this method).

Proof. Consider two adjacent vertices a and b in G. We show that a and b are in the same connected chordal graph H in the output iff we have $a - b$ in G^S. By transitivity it follows that two vertices are in the same output graph H iff there is an undirected path between them, which implies the first part of the statement.

If a and b are in different connected components output by Algorithm 5.1, then there was a step i in the algorithm at which a and b were in M_i 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 S (by Lemma 5.13 and Lemma 2.17), one time with $a \to b$, the other with $a \leftarrow b$. Hence, $a - b$ in G^S 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., $t(a) < t(b)$. We show by induction over i that $a \to b$ in G^S. We do this by proving that $a \to b$ in every graph in $\text{AMO}(G, S)$. For the start of the induction, observe that the vertices in S are not output at all and all edges from S to vertices in $V \setminus S$ are oriented towards the latter in every graph in $\text{AMO}(G, S)$. At step $i = t(a)$ when a was output, b was not in M_i. It follows that $P_i(a) \neq P_i(b)$. With $P_i(b) \setminus P_i(a) \neq \emptyset$ and as the algorithm produces a PEO by Lemma 5.13, it follows that $P_i(b) \subset P_i(a)$. Let c be in $P_i(a) \setminus P_i(b)$. By induction hypothesis, we have $c \to a$ in every graph in $\text{AMO}(G, S)$ as c is not output together with a (recall that i is the iteration when a is output, c has already been visited previously). This necessitates $a \to b$ for avoiding a v-structure (first Meek rule).

Similar to the standard MCS, the algorithm can be implemented in linear-time. □

Theorem 5.14 is an important result in its own right. Algorithm 5.1 may be used not only for computing $\mathcal{C}(G^S)$, but also G^s in linear time, which improves previous algorithms, such as an $O(n \cdot m)$ algorithm in [Talvitie and Koivisto, 2019] by a factor n. It can be generalized to find the interventional CPDAG based on given intervention results (see Chapter 6 for a discussion).

For now, we focus on the structural properties of AMOs revealed by Algorithm 5.1, which allow us to conclude (i) that the components in $\mathcal{C}(G^S)$ are chordal and (ii) can be oriented independently. The first fact can be easily seen as, by Algorithm 5.1, the connected 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 connected component have the same parent set in G^S, which ensures that any AMO of the component will not create a new v-structure or cycle in G^K. Crucially, this paves the way towards a recursive formulation of #AMO based on picking a clique as source.

Corollary 5.15. *Let G be a chordal graph, S a clique and σ a permutation of S.*

1. *The connected components $\mathcal{C}(G^S)$ are induced subgraphs of G and hence chordal graphs.*
2. *Let adjacent x, y in G be in different connected components of G^S and $t(x) < t(y)$. Then, $x \to y$ is an edge in G^S.*
3. $P_{t(v)}(v) = Pa_{G^S}(v)$.
4. *For adjacent a, b in the same undirected component of G^S, we have that $P_{t(a)} = P_{t(b)}$.*
5. $|AMO(G, \sigma)| = \prod_{H \in \mathcal{C}(G^\sigma)} |AMO(H)|$ *and*
6. $|AMO(G, S)| = |S|! \times \prod_{H \in \mathcal{C}(G^S)} |AMO(H)|$.

Proof. We show the statements one-by-one:

1. Follows immediately from Theorem 5.14 and the fact that Algorithm 5.1 returns induced subgraphs (chordality is a hereditary graph property as is extendability, see Lemma 3.10).

2. Shown in the proof of Theorem 5.14.

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

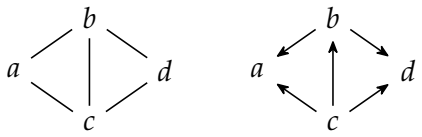
4. As a and b are output in the same iteration $i = t(a) = t(b)$, they both are in M_i, and could both have been picked as vertex x. However, if $P_i(a) \setminus P_i(b) \neq \emptyset$ or $P_i(b) \setminus P_i(a) \neq \emptyset$ the algorithm would not produce a PEO (after the choice of either a or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

5. By Lemma 5.11, we have that $|\mathrm{AMO}(G, \sigma)| = |\mathrm{EXT}(G^\sigma)|$. Moreover, by (1.) the connected components in $\mathcal{C}(G^\sigma)$ 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 5.11 and Lemma 5.12 we have that $|\mathrm{AMO}(G, \pi_S)| = |\mathrm{AMO}(G, \pi'_S)| = |\mathrm{EXT}(G^{\pi_S})| = |\mathrm{EXT}(G^S)|$ for any π_S, π'_S. As there are $|S|!$ many permutations, which all lead to different AMOs, and combined with (5.) we arrive at the stated formula. □

Based on item (6.) of Corollary 5.15 and justified by Lemma 5.8, we would like to count the AMOs of a connected 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 graphs $H \in \mathcal{C}(G^K)$. Due to Lemma 5.8, 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 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 5.16. *Consider the following chordal graph (left) with maximal cliques $K_1 = \{a, b, c\}$ and $K_2 = \{b, c, d\}$. A possible AMO of the graph is shown on the right.*



The AMO has two topological orderings: $\tau_1 = (c, b, a, d)$ and $\tau_2 = (c, b, d, a)$ 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 (c, b) as common prefix and $K_1 \cap K_2 = \{b, c\}$ is a minimal separator of the graph – a fact that we will use in the following. ◇

Towards characterizing this phenomenon, we start with the following straightforward lemma (recall that we use π_S to denote a permutation of the set of vertices S).

Lemma 5.17. *Let G be a connected chordal graph, D be an AMO of G, π_S be the prefix of some topological ordering $\tau \in \text{top}(D)$ and $S' \supseteq S$ a clique in G. Then, π_S is a prefix of every topological ordering in $\text{top}(D, S')$.*

Proof. Consider, for the sake of contradiction, there exists a topological ordering $\tau' \in \text{top}(D, S')$ not having π_S as prefix. If the vertices in S would be ordered in τ' according to a permutation $\pi'_S \neq \pi_S$, then it cannot be a topological ordering of D. It follows that there exist vertices $v \in S' \setminus S$ and $s \in S$ such that v comes before s in τ' implying the edge $v \rightarrow s$ in D. However, τ is a topological ordering of D with S as prefix, which makes an edge $v \rightarrow s$, generally for any v not in S, impossible. □

A trivial special case of this lemma is that, for a clique S, all topological orderings in $\text{top}(D, S)$ start with the same permutation of S. Building on this we show that (with $\alpha[1:k]$ denoting the first k elements of a sequence α):

Lemma 5.18. *Let G be a connected chordal graph and $R \subseteq \Pi(G)$ a set of maximal cliques with AMO $D \in \bigcap_{K \in R} AMO(G, K)$. Then,*

1. *$S \coloneqq \bigcap_{K \in R} K$ is a minimal separator or maximal clique of G and*
2. *a permutation π_S is the longest common prefix of $\bigcup_{K \in R} \text{top}(D, K)[1 : |K|]$.*

Proof. We show the statement by induction over the cardinality of R. If R is a singleton set, (1.) is obvious and (2.) follows from Lemma 5.17. Assume the induction hypothesis that $S = \bigcap_{K \in R} K$ is a minimal separator or maximal clique of G and longest common prefix of $\bigcup_{K \in R} \text{top}(D, K)[1 : |K|]$. Consider now $R' = R \cup \{K'\}$ and define $S' \coloneqq \bigcap_{K \in R'} K = S \cap K'$ by associativity. We show that the statement holds for R' by distinguishing two cases.

If $S' = S$, then (1.) S' is a minimal separator by induction hypothesis and (2.) longest common prefix of $\bigcup_{K \in R'} \text{top}(D, K)[1 : |K|]$ by induction hypothesis and Lemma 5.17.

If $S' \subset S$, then there exist by induction hypothesis $v \in S \setminus S'$, which is at position $|S'| + 1$ in every topological ordering in $\bigcup_{K \in R} \text{top}(D, K)$. Consider $w \in K' \setminus S'$ which is at position $|S'| + 1$ in every topological ordering in $\text{top}(D, K')$ by Lemma 5.17. By Lemma 3.10, $D[V \setminus S']$ is an AMO of $G[V \setminus S']$. By Lemma 5.4, v and w cannot be in the same connected component in $G[V \setminus S']$. Hence, S' is an $v - w$ separator and, as v and w are fully connected to S' due to being in cliques which are supersets of S', it is a minimal one. Moreover, by induction hypothesis and Lemma 5.17, we have that S' is a prefix of every topological ordering of $\text{top}(D, K')$ and due to existence of v and w the longest common prefix of $\bigcup_{K \in R'} \text{top}(D, K)[1 : |K|]$. □

This insight allows us to formulate a novel recursive formula for #AMO. We will further refine it below, however, it already illustrates the kind of recursive formula we are aiming for and yields a polynomial-time algorithm. It relies on the combinatorial function ϕ.

Definition 5.19 ($\phi(S)$)**.** *Let G be a connected chordal graph and $S \in \Sigma(G) \cup \Pi(G)$. We define $\phi(S)$ as the set of permutations of S, which do not have a permutation $\pi_{S'}$ as a prefix for a minimal separator $S' \subset S$.*

The goal is to count an AMO D at a single minimal separator or maximal clique, as *all* topological orderings $\text{top}(D)$ share the same prefix which is the former or the latter by Lemma 5.18.

Proposition 5.20. *Let G be a connected chordal graph. Then:*

$$|AMO(G)| = \sum_{S \in \Sigma(G) \cup \Pi(G)} |\phi(S)| \times \prod_{H \in \mathcal{C}(G^S)} |AMO(H)|.$$

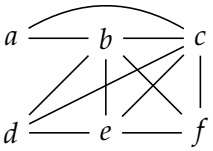
Proof. By Corollary 5.15, we have that

$$\begin{aligned} |\text{AMO}(G)| &= \sum_{S \in \Sigma(G) \cup \Pi(G)} |\phi(S)| \times \prod_{H \in \mathcal{C}(G^S)} |\text{AMO}(H)| \\ &= \sum_{S \in \Sigma(G) \cup \Pi(G)} \frac{|\phi(S)|}{|S|!} \times |\text{AMO}(G, S)|. \end{aligned}$$

Every term of the sum counts the number of AMOs which can be represented by a topological ordering with $\pi_S \in \phi(S)$ as prefix. We argue that every AMO D is counted at exactly one term.

Let $S \in \Sigma(G) \cup \Pi(G)$ be the smallest common prefix of all topological orderings in $\text{top}(D)$, which is a minimal separator or maximal clique of G. This set is well-defined by Lemma 5.18 and Lemma 5.8. First observe that, by the minimality of S, D is counted at the term for S: There is no other prefix $S' \subset S$ of the topological orderings with $S' \in \Sigma(G) \cup \Pi(G)$, which implies that the permutation of S corresponding to the prefix is in $\phi(S)$. Conversely, S is the only term in the sum at which D is counted, as for any larger S' with $S \subset S'$ that is a prefix of some $\tau \in \text{top}(D)$, we have that any permutation of S' with S as prefix is not in $\phi(S)$ and, by Lemma 5.18 and the construction of S, τ contains such a prefix π_S. □

Example 5.21. *We illustrate the formula in Proposition 5.20 for the following graph:*

$G =$ 

$$\Sigma(G) = \{\{b, c\}, \{b, c, e\}\}$$
$$\Pi(G) = \{\{a, b, c\}, \{b, c, d, e\}, \{b, c, e, f\}\}$$

Let $H \coloneqq G[\{d, e, f\}]$ be the induced subgraph $d - e - f$. For the sake of readability, we omit terms in the product $\prod_{H \in \mathcal{C}(G^S)} |AMO(H)|$ that are equal to one.

$$\begin{aligned} |AMO(G)| &= |\phi(\{b, c\})| \cdot |AMO(H)| + |\phi(\{b, c, e\})| + |\phi(\{a, b, c\})| \cdot |AMO(H)| + \\ &\quad |\phi(\{b, c, d, e\})| + |\phi(\{b, c, e, f\})| \\ &= 2 \cdot 3 + 4 \cdot 1 + 4 \cdot 3 + 16 \cdot 1 + 16 \cdot 1 = 54. \end{aligned}$$

We discuss how ϕ can be computed efficiently further below. ◇

This function can already be implemented to yield polynomial-time.[3] However, we will further refine our approach summing only over the maximal cliques. While this will lead to a more intricate analysis, the algorithm itself can be implemented easily and efficiently. It will also lend itself well to efficient uniform sampling of AMOs (see Section 6.1) due to the restrictions we impose on ϕ. We aim for a formula of the form

$$|\text{AMO}(G)| = \sum_{K \in \Pi(G)} |\phi_{T^R}(K)| \prod_{H \in \mathcal{C}(G^K)} |\text{AMO}(H)|.$$

[3] In [Wienöbst et al., 2021b] and [Wienöbst et al., 2023], it is stated that it is unclear whether $\phi(S)$ as defined above can be computed efficiently. Indeed, it *can*, by using very similar techniques as in Theorem 5.37. We give a remark on this below.

Here, each AMO D is counted at one maximal clique K – in contrast to a maximal clique *or* minimal separator above. For dealing with overcounting, function ϕ_{T^R} is defined with regard to a rooted clique tree T^R (the formal definition of T^R was given in Chapter 2). To characterize at which clique the AMO D is counted needs the following machinery which has *S-flowers*, defined with regard to a minimal separator S, at its core. In the following, we denote $\{K \mid K \in \Pi(G) \wedge S \subseteq K\}$ by $\Pi^S(G)$ thus generalizing $\Pi^v(G)$ defined in Chapter 2.

Definition 5.22 (S-flower)**.** *Let G be a connected chordal graph and S a minimal separator. An* S-flower *is a maximal set $F \subseteq \Pi^S(G)$ 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. For $F \in \mathcal{B}(S)$, we denote the set of vertices in any clique in F by $V_F := \bigcup_{K \in F} K$.*

We start with a few immediate facts about S-flowers.

Lemma 5.23. *Let G be a connected chordal graph and $S \in \Sigma(G)$.*

1. *$\mathcal{B}(S)$ is a partition of the set of cliques $\Pi^S(G)$,*
2. *$T^R[F]$ for $F \in \mathcal{B}(S)$ is a connected subtree of $T^R[\Pi^S(G)]$,*
3. *$\mathcal{B}(S)$ partitions $T^R[\Pi^S(G)]$ in connected subtrees,*
4. *V_F for $F \in \mathcal{B}(S)$ is a subset of the vertices of a connected component in $G[V \setminus S]$ and*
5. *for $K_1 \in F_1$ and $K_2 \in F_2$ for distinct $F_1, F_2 \in \mathcal{B}(S)$ it holds $K_1 \cap K_2 = S$.*

Proof. We show the statements one-by-one.

1. It follows directly from Definition 5.22 that every clique is in at least one flower. Assume, for a sake of contradiction, clique K is in $F_1 \in \mathcal{B}(S)$ *and* $F_2 \in \mathcal{B}(S)$ with $F_1 \neq F_2$. Then, as every clique $A \in F_1$ and $B \in F_2$ is connected to K in $G[V \setminus S]$ by definition, we have that A is connected to B in $G[V \setminus S]$ by transitivity of connectedness. This violates the maximality of F_1 and F_2.
2. 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 $T^R[\Pi^S(G)]$ is connected by definition of clique trees, 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.
3. Immediately follows from items (1.) and (2.).
4. By Definition 5.22 the vertices V_F are connected in $G[V \setminus S]$.
5. Let $S' := K \cap K'$. Every $s \in S$ is in S' by Definition 5.22. Conversely, $s \notin S$ in S' would contradict item (4.), which implies $S = S'$. □

Items (1.) to (3.) allow defining a partial order over S-flowers with regard to a rooted clique-tree T^R.

Definition 5.24 ($\preceq^S_{T^R}$)**.** *Let G be a connected chordal graph, T^R be a rooted clique-tree of G, $S \in \Sigma(G)$ and $F_1, F_2 \in \mathcal{B}(S)$. Then, $F_1 \preceq^S_{T^R} F_2$ if there exists $K_1 \in F_1$ such that for every $K_2 \in F_2$, it holds that K_1 is on the unique path from R to K_2.*

Lemma 5.25. *The relation $\preceq^S_{T^R}$ defines a partial order which has a least element.*

Proof. It is easy to see that $\preceq^S_{T^R}$ defines a partial order. To see that it has a least element, observe that $T^R[\Pi^S(G)]$ has a clique K closest to R. It is on the path from R to every clique in $T^R[\Pi^S(G)]$ and hence F with $K \in F$ is the least flower. □

Our goal is to associate an AMO D with a single maximal clique K for which $D \in \mathrm{AMO}(G, K)$ and to count D at precisely this clique. For this, we need the following observations:

Lemma 5.26. *Let G be a connected chordal graph, $S \in \Sigma(G)$, $D \in AMO(G, S)$ and $F \in \mathcal{B}(S)$.*

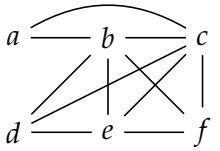
1. *There exists $K \in F$ such that $D \in AMO(G, K)$ and*
2. *for $K, K' \in F$ with $D \in AMO(G, K)$ and $D \in AMO(G, K')$, it holds that $S' := K \cap K'$ satisfies $S \subset S'$.*

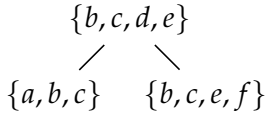
Proof.

1. By definition, in AMO D every edge between $u \in S$ and $v \notin S$ is directed $u \to v$. The order of vertices in different connected components of $G[V \setminus S]$ in a topological ordering of D can be chosen arbitrarily. Let H be the connected component of $G[V \setminus S]$ that F is in (by Lemma 5.23 item (4.)) and consider w.l.o.g. that vertices V_H come immediately after S in topological ordering τ. By Lemma 5.8, there exists a topological ordering of $D[S \cup V_H]$ (which is an AMO of $G[S \cup V_H]$ by Lemma 3.10) starting with a maximal clique, completing the proof.
2. By Lemma 5.18, it holds that (i) S' is a minimal separator and (ii) there exists $\pi_{S'}$ which is longest common prefix of the topological orderings in $\mathrm{top}(D, K) \cup \mathrm{top}(D, K')$. By (i), K and K' are in different S' flowers. It follows that $S' \not\subseteq S$. Because $D \in \mathrm{AMO}(G, S)$, we have, by (ii), $S \subset K$ as well as $S \subset K'$ and Lemma 5.17 that there exists a prefix π_S, which is a prefix of any topological ordering in $\mathrm{top}(D, K) \cup \mathrm{top}(D, K')$. Hence, S and S' have an inclusion relation, which can only be $S \subset S'$ by the arguments above. □

These results form the basis of our new approach. Consider AMO D and the set $\mathrm{top}(D)$ of topological orderings representing D. Then, by Lemma 5.18, those have a common prefix S which is a minimal separator (or maximal clique). Now the partial order $\preceq^S_{T^R}$ over the S-flowers in bouquet $\mathcal{B}(S)$ has a least element F. By Lemma 5.26 item (1.), F contains a maximal clique K such that $D \in \mathrm{AMO}(G, K)$. If this clique is unique, then count D at K. If not, then the cliques $K \in F$ with $D \in \mathrm{AMO}(G, K)$ have a common prefix, which is another minimal separator S' (with $S \subsetneq S'$), again by Lemma 5.18. Here, a recursive argument applies, which ultimately leads to a unique clique at which we aim to count D.

Example 5.27. *We illustrate this for the example graph from Example 5.21 with a clique-tree representation on the right:*

$G =$ 

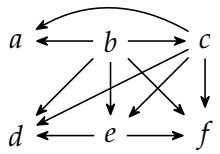
$T^{\{b,c,d,e\}} =$ 

Below, the clique-starting topological orderings of DAG D (right) are shown on the left. They all have a common prefix (b, c), which is a minimal separator. It has two flowers, the one with cliques $\{b, c, d, e\}$ and $\{b, c, e, f\}$ is lesser than $\{\{a, b, c\}\}$ with regard to $\preceq^{\{b,c\}}_{T^R}$. In this lesser flower,

again the common prefix is a minimal separator $\{b, c, e\}$. Repeating the procedure ultimately yields $\{b, c, d, e\}$ as the clique at which we aim to count D.

$\mathrm{top}(D)$

$b,$	$c,$	$a,$	$e,$	$d,$	$f,$
$b,$	$c,$	$a,$	$e,$	$f,$	$d,$
$b,$	$c,$	$e,$	$d,$	$f,$	$a,$
$b,$	$c,$	$e,$	$d,$	$a,$	$f,$
$b,$	$c,$	$e,$	$f,$	$d,$	$a,$
$b,$	$c,$	$e,$	$f,$	$a,$	$d,$

$D =$ 

$$\{\{b,c,d,e\},\{b,c,e,f\}\} \prec_{T^R}^{\{b,c\}} \{\{a,b,c\}\}$$

$$\{\{b,c,d,e\}\} \prec_{T^R}^{\{b,c,e\}} \{\{b,c,e,f\}\}$$

In the following, we formalize this argument and, building on it, we derive a new counting procedure. ◇

We proceed in a non-constructive way showing that there exists a partial order over cliques K with $D \in \mathrm{AMO}(G, K)$ with a least element. It will neither be necessary nor efficient to explicitly construct this clique for each AMO, instead our counting algorithm will implicitly consider an AMO only at the least clique.

Definition 5.28 ($\preceq_{T^R}^{D}$)**.** *Let G be a connected chordal graph, T^R a rooted clique-tree of G, D an AMO of G and distinct $K_1, K_2 \in \Pi(G)$ with $D \in AMO(G, K_1) \cap AMO(G, K_2)$. Moreover, $S = K_1 \cap K_2$ and $K_1 \in F_1$, $K_2 \in F_2$ with $F_1, F_2 \in \mathcal{B}(S)$. Then, $K_1 \preceq_{T^R}^{D} K_2$ if $F_1 \preceq_{T^R}^{S} F_2$.*

This definition assumes $K_1 \neq K_2$, in case the cliques coincide, we define $K_1 \preceq_{T^R}^{D} K_2$.

Lemma 5.29. *The relation $\preceq_{T^R}^{D}$ defines a partial order which has a least element.*

Proof. The crucial part of showing that $\preceq_{T^R}^{D}$ is a partial order is transitivity. Assume we have $K_1 \preceq_{T^R}^{D} K_2$ and $K_2 \preceq_{T^R}^{D} K_3$. We show that $K_1 \preceq_{T^R}^{D} K_3$ holds as well. For this, observe that $K_1 \cap K_2 = S$ and $K_2 \cap K_3 = S'$ satisfy either (i) $S = S'$, (ii) $S \subset S'$ or (iii) $S \supset S'$.

To see this, let S be the common prefix of any topological ordering $\mathrm{top}(D, K_1)$ and $\mathrm{top}(D, K_2)$ by Lemma 5.18 and S' the common prefix of any ordering in $\mathrm{top}(D, K_2)$ and $\mathrm{top}(D, K_3)$. Then, the orderings in $\mathrm{top}(D, K_2)$ start with S *and* S', which implies the inclusion relations above.

We consider case (ii) first and denote $K_1 \cap K_3 = S''$. With $S \subset S'$, we have that $S'' = S$ because K_2 and K_3 are in the same S-flower (by contraposition of Lemma 5.23 item (5.)), which is by assumption lesser than the one K_1 is in. Here, and by symmetry for (iii), transitivity follows. Consider now case (i). By contraposition of Lemma 5.26 item (2.), K_1, K_2 and K_3 are in different S-flowers with $F_1 \ni K_1$ being the greatest and $F_3 \ni K_3$ being the least by assumption, again implying transitivity.

It remains to show that the partial order has a least element. Any partial order over a finite (non-empty) set has a minimal element, it remains to show that it is unique. For the sake of contradiction, assume there are two distinct minimal elements K_1 and K_2. Then, $S = K_1 \cap K_2$ has a least S-flower F by Lemma 5.25, which contains a clique K_3 with $\mathrm{top}(D, K_3) \neq \emptyset$ by Lemma 5.26 item (1.). K_1 and K_2 are in different incomparable S-flowers by contraposition of Lemma 5.26 item (2.) and under the assumption that they are minimal elements. In particular, this implies that K_1 and K_2 are not in F and thus they are greater than K_3. A contradiction to the minimality of K_1 and K_2. □

To make use of these results, it is necessary to design an algorithm, which (implicitly) counts D exactly at the least clique with regard to $\preceq^D_{T^R}$ for some rooted clique-tree T^R.

Consider a clique K which is not least element with regard to AMO D. Then, there is at least one minimal separator S, such that K is not in the least S-flower. Conversely, this is not the case if K is least element. In the following we exploit that S is the intersection of two adjacent cliques on the R-K path. Before proving this formally below, we introduce the corresponding counting function $\phi_{T^R}(K)$.

Definition 5.30 ($\phi_{T^R}(K)$)**.** *Let G be a connected chordal graph and T^R be a rooted clique-tree of G. Then, $\phi_{T^R}(K)$, for $K \in \Pi(G)$, describes the set of permutations of K, which do not have a separator $S = K_i \cap K_j$ as prefix, with K_i and K_j being adjacent vertices on the path from R to K.*

Such a separator S is called a *forbidden prefix* of K.

Lemma 5.31. *Let T^R be a rooted clique-tree of G, $K \in \Pi(G)$, $D \in AMO(G, \pi_K)$. Then, K is least clique with regard to $\preceq^D_{T^R}$ if, and only if, $\pi_K \in \phi_{T^R}(K)$.*

Proof. We show two directions. Assume K is not least clique with regard to $\preceq^D_{T^R}$. Then, there exists a separator $S \subset K$ such that K is in S-flower F distinct from the least S-flower F'. By Definition 5.24, this means that there is a clique $K' \in F'$ on the unique path from R to K in T^R. By Lemma 5.23 item (3.), this necessitates two adjacent cliques on the path from R to K, which are in different S-flowers (more precisely on the $K' - K$ subpath). By Lemma 5.23 item (5.), these cliques have intersection S. In turn, this means that $S \subseteq K$ is a forbidden prefix of K with regard to $\phi_{T^R}(K)$ and that π_K is not in $\phi_{T^R}(K)$.

In the converse, assume that $\pi_K \notin \phi_{T^R}(K)$. Then, there exists a minimal separator $S \subseteq K$ at the beginning of π_K such that there is an edge on the path from K to R which is associated with S. This implies that K is not in the least S-flower and that, by Lemma 5.26 item (1.), the least S-flower contains a clique, which can represent D and is thus lesser then K. □

The following recursive formula immediately follows:

Proposition 5.32. *Let G be a connected chordal graph and T^R be a rooted clique tree of G. Then*

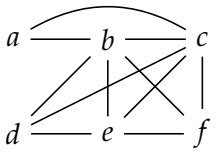
$$|AMO(G)| = \sum_{K \in \Pi(G)} |\phi_{T^R}(K)| \times \prod_{H \in \mathcal{C}(G^K)} |AMO(H)|.$$

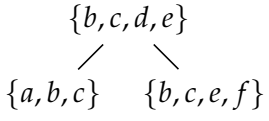
Proof. Analogously to the proof of Proposition 5.20, by Corollary 5.15, we have that

$$\begin{aligned} |\mathrm{AMO}(G)| &= \sum_{K \in \Pi(G)} |\phi_{T^R}(K)| \times \prod_{H \in \mathcal{C}(G^K)} |\mathrm{AMO}(H)| \\ &= \sum_{K \in \Pi(G)} \frac{|\phi_{T^R}(K)|}{|K|!} \times |\mathrm{AMO}(G, K)|. \end{aligned}$$

Thus, it remains to argue that AMO D is counted at precisely one term of the sum. This directly follows from Lemma 5.31. □

Example 5.33. *We illustrate the formula in Proposition 5.32 for the graph from above:*

$G =$ 

$T^{\{b,c,d,e\}} =$ 

As before, let $H \coloneqq G[\{d,e,f\}]$ be the induced subgraph $d-e-f$. For the sake of readability, we again omit terms in the product $\prod_{H\in\mathcal{C}(G^K)} |AMO(H)|$ that are equal to one.

$$\begin{aligned}|AMO(G)| &= |\phi_{T^R}(\{a,b,c\})| \cdot |AMO(H)| \; + |\phi_{T^R}(\{b,c,d,e\})| + |\phi_{T^R}(\{b,c,e,f\})| \\ &= 4 \cdot 3 + 24 \cdot 1 + 18 \cdot 1 = 54.\end{aligned}$$

In particular, the six permutations of $\{b,c,e,f\}$ which are forbidden *are of the form $\pi_{\{b,c,e\}} + \pi_{\{f\}}$. Below, we discuss how ϕ_{T^R} can be efficiently evaluated in general.* ◇

Algorithm 5.2: Computing $|\mathrm{AMO}(G)|$ in polynomial-time with Clique-Picking.

input : A connected chordal $G = (V, E)$.
output: $|\mathrm{AMO}(G)|$.

```
1  function count(G, memo)
2      if G ∈ memo then return memo[G]
3      T^R := a rooted clique tree of G
4      // Calculate |AMO(G)| in variable sum.
5      sum := 0
6      foreach K ∈ Π(G) do
7          // Calculate ∏_{H∈C(G^K)} |AMO(H)| in variable prod.
8          // C(G^K) can be computed by Alg. 5.1 in time O(|V| + |E|).
9          prod := 1
10         foreach H ∈ C(G^K) do
11             prod := prod · count(H, memo)
12         end
13         // φ_{T^R}(K) can be computed by Alg. 5.3 in time O(|V| + |E|).
14         sum := sum + |φ_{T^R}(K)| · prod
15     end
16     memo[G] = sum
17     return sum
18 end
19 memo := empty hash-table
20 return count(G, memo)
```

Algorithm 5.2 provides pseudocode for evaluating this recursive formula with memoization (that is storing the results for each function call to avoid repetitive calculations).

Theorem 5.34. *Given a connected chordal graph G, Algorithm 5.2 returns the number of AMOs of G.*

Proof. Follows from Proposition 5.32 and the fact that the algorithm implements the stated recursive formula. □

It remains to show that this approach yields a polynomial-time algorithm: For this, we show that (i) the right side of the formula in Proposition 5.32 can be evaluated in polynomial-time, assuming $|\mathrm{AMO}(H)|$ was already recursively computed. In particular we need to show how $|\phi_{T^R}(K)|$ can be calculated in polynomial time, and (ii) that the recursion tree contains only polynomially-many distinct elements. In combination, this yields a polynomial-time algorithm for $\#\mathrm{AMO}(G)$.

We begin with (i) and efficiently calculating $|\phi_{T^R}(K)|$. For this, we introduce the function $\rho(X)$ for strictly increasing sequence X.

Definition 5.35 ($\rho(X)$)**.** *Let $X = \{x_1, x_2, \dots\}$ be a strictly increasing sequence. Then, define $\rho(X)$ to be the set of permutations of $\{1, 2, \dots, x_k\}$ which do not contain $\{1, 2, \dots, x_i\}$ for $i < k$ as prefix.*

This naturally generalizes the notion of *irreducible permutations*, which on $\{1, 2, \dots, n\}$ coincide with $\rho((1, 2, \dots, n))$. It is rather easy to see that the size of $\rho(X)$, for an appropriate X, can be associated with the size of $\phi_{T^R}(K)$.

Lemma 5.36. *Let T^R be a rooted clique-tree of G and $K \in \Pi(G)$. Then, $|\phi_{T^R}(K)| = |\rho(X + (|K|))|$ with sequence X containing elements $|S|$ for sets $S = K_i \cap K_j$ with K_i and K_j adjacent on the path from K to R, which satisfy $S \subseteq K$, ordered by decreasing distance from K on the path.*

Proof. First, observe that only separators $S = K_i \cap K_j$ with $S \subseteq K$ can be forbidden prefixes of any permutation of K with regard to $\phi_{T^R}(K)$. Moreover, duplicates S can be removed. The remaining separators S form a strictly increasing sequence when ordered by decreasing distance from K. To see this, observe that if S_1 further away from K than S_2 would contain a $v \in S_1 \setminus S_2$, it would hold that $T^R[\Pi^v(G)]$ is not connected violating the definition of clique-trees. Finally, there is an bijection from the vertices in K to $\{1, 2, \dots, |K|\}$ such that any S in $\mathcal{S}$ can be mapped to $\{1, 2, \dots, |S|\}$. The statement immediately follows. □

We now show that $|\rho(X)|$ can be computed efficiently.

Theorem 5.37. *Let $X = (x_1, x_2, \dots, x_k)$ be a strictly increasing sequence. Then, $|\rho(X)|$ can be computed in time $O(k^2)$.*

Proof. It holds for $X = (x_1, \dots, x_k)$ that

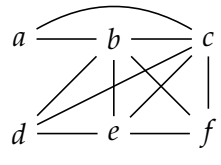
$$|\rho(X)| = x_k! - \sum_{i=1}^{k} (x_k - x_i)! \cdot |\rho((x_1, \dots, x_i))|.$$

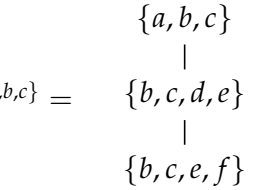
To see this let π be a permutation of $\{1, \dots, x_k\}$, such that $\{1, \dots, x_i\}$ for $i < k$ is its shortest prefix of this form. Clearly, this permutation is counted precisely at term i of the sum and thus subtracted once from $x_k!$. By induction, the statement follows. □

We are now able to give an algorithm for computing $|\phi_{T^R}(K)|$. The correctness and run-time of the algorithm follows immediately from Theorem 5.37 and its proof.

Corollary 5.38. *Algorithm 5.3 computes $|\phi_{T^R}(K)|$ in time $O(|V_{T^R}| + |K|^2)$.*

Example 5.39. *As a non-trivial example for the computation of the size of ϕ_{T^R} consider our graph from the examples above, this time with the clique-tree rooted at maximal clique $\{a, b, c\}$:*

$G =$ 

$T^{\{a,b,c\}} =$ 

We compute the cardinality of $\phi_{T^R}(\{b, c, e, f\})$. This sets consists, by definition, of all permutations of $\{b, c, e, f\}$ which start neither with $\{b, c\}$ (the separator associated to the first edge on the path from the root to $\{b, c, e, f\}$) nor with $\{b, c, e\}$ (the separator associated to the second such

Algorithm 5.3: Computing $|\phi_{T^R}(K)|$ in polynomial-time.

input : A rooted clique-tree T^R and a maximal clique $K \in V_{T^R}$.
output: $|\phi_{T^R}(K)|$.

```
1  function rho(X = (x_1, ..., x_k))
2      for i := 1 to k do
3          ρ_i := x_i!
4          for j := 1 to i − 1 do ρ_i := ρ_i − (x_i − x_j)! · ρ_j
5      end
6      return ρ_k
7  end
8  X := (K)
9  foreach edge K_i − K_j on the path from K to R in T^R in increasing distance from K do
10     S := K_i ∩ K_j
11     if first(X) ≠ |S| and S ⊆ K then
12         X := (|S|) + X
13     end
14     if S ∩ K = ∅ then break
15 end
16 return rho(X)
```

edge). The desired quantity can also be expressed as $|\rho((2,3,4))|$*, where* 2*,* 3*, and* 4 *denote the cardinality of* $\{b,c\}$*,* $\{b,c,e\}$*, and* $\{b,c,e,f\}$*. We have that:*

$$\begin{aligned}|\rho((2,3,4))| &= 4! - 2! \cdot |\rho((2))| - 1! \cdot |\rho((2,3))| \\ &= 4! - 2! \cdot 2! - 1! \cdot (3! - 1! \cdot |\rho((2))|) \\ &= 4! - 2! \cdot 2! - 1! \cdot (3! - 1! \cdot 2!) \\ &= 24 - 4 - 4 = 16.\end{aligned}$$

To complete the example, we show that rooting the clique-tree at clique $\{a,b,c\}$ *again yields* $|AMO(G)| = 54$ *through the computation below. As in the examples above, we have* $H := G[\{d,e,f\}]$ *and omit trivial terms in the product* $\prod_{H \in \mathcal{C}(G^S)} |AMO(H)|$*.*

$$\begin{aligned}|AMO(G)| &= |\phi_{T^R}(\{a,b,c\})| \cdot |AMO(H)| \; + |\phi_{T^R}(\{b,c,d,e\})| + |\phi_{T^R}(\{b,c,e,f\})| \\ &= 6 \cdot 3 + 20 \cdot 1 + 16 \cdot 1 = 54.\end{aligned}$$

Hence, the choice of the clique-tree and its root has an influence on the performed arithmetic operations, however not on the overall result and the recursive subproblems. ◇

We note that the restriction to strictly increasing sequences is not necessary for efficiently computing the size of ϕ. Let S be a set of, e.g., vertices, $\mathcal{R}$ a collection of subsets of S and $|\phi(S,\mathcal{R})|$ denote the number of permutations of S which do not have π_R for any $R \in \mathcal{R}$ as prefix. Similar to the formula in the proof of Theorem 5.37, it holds that

$$|\phi(S,\mathcal{R})| = |S|! - \sum_{R \in \mathcal{R}} (|S| - |R|)! \times |\phi(R, \{R' \in \mathcal{R} \mid R' \subset R\})|.$$

It is easy to see that this formula, combined with the analysis below, implies that Proposition 5.20 would also lead to a polynomial-time algorithm. However, we will focus solely on the formula in Proposition 5.32 in the following.

It remains to show that relying on this approach, there are at most polynomially-many distinct recursive calls of $|\mathrm{AMO}(G)|$. For this, recall that $P_{t(v)}(v)$ are the neighbors of v, which are already visited, i.e. which are in $\tau_{t(v)}$, at step $t(v)$ of the algorithm, that is when v is handled. By Corollary 5.15 item (4.), we have that $P_{\tau(v)}(v)$ is identical for all $v \in V_H$, where $H \in \mathcal{C}(G^K)$ for some clique K. Hence, slightly abusing notation, we write $P_{t(H)}(H)$ and $\tau_{t(H)}(H)$ instead of $P_{t(v)}(v)$ and $\tau_{t(v)}(v)$ for any vertex v in V_H.

Lemma 5.40. *Let G be a connected chordal graph and $H \in \mathcal{C}(G^K)$. Then, $P_{t(H)}(H)$ separates V_H from $W = \tau_{t(H)} \setminus P_{t(H)}(H)$ and is a minimal separator of G.*

Proof. The set $P_{t(H)}(H)$ is a proper subset of all visited vertices at step $t(H)$ (as V_H is not part of the *maximal* clique K Algorithm 5.1 starts with) implying that W is non-empty. Since $P_{t(H)}(H)$ contains all visited neighbors of H, it separates V_H from W. To see this, assume for sake of contradiction that there is a path from $v \in V_H$ to $w \in W$ without a vertex in $P_{t(H)}(H)$. Consider the shortest such path and let y be the first vertex with successor z preceding it in the vertex ordering produced by Algorithm 5.1: $v - \cdots - x - y - z - \cdots - w \in W$. 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 $P_{t(H)}(H)$ is indeed a separator.

Let us now show minimality. We prove that there is a vertex in W, which is a neighbor of all vertices in $P_{t(H)}(H)$. Consider the vertex in $P_{t(H)}(H)$, 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 (namely, $P_{t(H)}(H) \setminus \{p\}$), meaning that p would have the same number of already visited neighbors. It would follow that in line 13 of Algorithm 5.1 either p and the vertices in H are appended to T_i when p is visited or were already appended to T_j previously (e.g., for $j < i$). 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 $P_{t(H)}(H)$ 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 $P_{t(H)}(H)$. As both x and y are fully connected to $P_{t(H)}(H)$, it follows that this set is also a *minimal* $x - y$ separator. □

Lemma 5.41. *Let G be a connected chordal graph for which the number of AMOs is computed with the function* `count` *in Algorithm 5.2. Let H be any chordal graph for which* `count` *is called in the recursion (for $H \neq G$). Then $V_H = V_F \setminus S$ for some S-flower F in G with $S \in \Sigma(G)$.*

Proof. Let S_H be the union of all sets $P_{t(\tilde{G})}(\tilde{G})$ for $\tilde{G}$ on the recursive call stack from the input graph G to currently considered subgraph H. We define $P_{t(G)}(G) = \emptyset$ for convenience. Let $H \neq G$, we show by induction that (i) S_H is a minimal separator, (ii) S_H is fully connected to H, and (iii) $V_H = V_F \setminus S_H$ for some S_H-flower F.

In the base case, $H \in \mathcal{C}(G^K)$. By Lemma 5.40, S_H is a minimal separator in G, which is by definition connected to all vertices in H. Hence, as H is connected, $V_H \subseteq V_F \setminus S_H$ holds for an S_H-flower F. We show the equality by contradiction. Assume there is a vertex $v \in V_F \setminus S$ 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 5.40). 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, $G' = V_{F'} \setminus S_{G'}$ for some F'-flower of $S_{G'}$. Now, $P_{t(H)}(H)$ is by Lemma 5.40 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 $S_H = S_{G'} \cup P_{t(H)}(H)$ is a minimal x-y separator in G. Furthermore, S_H is fully connected to H and it can be easily seen that $H \subseteq V_F \setminus S_H$. To show equality, observe that every vertex v in $V_F \setminus S_H$ is in G' (if it is not separated from V_H by S_H, it is clearly not separated from V_H in $S_{G'}$). Thus, the same argument as in the base case applies and the statement follows. □

Finally, we are able to show:

Proposition 5.42. *Let G be a connected chordal graph. The number of distinct recursive calls explored by* `count` *is bounded by* $2|\Pi(G)| - 1$.

Proof. By Lemma 5.41, 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 connected to 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 Lemma 5.23 item (3.) the flowers partition the bouquet and by item (5.) the intersection of cliques from two distinct S-flowers is 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 distinct recursive calls, we additionally take into account the input graph and obtain as bound $2(|\Pi(G)| - 1) + 1 = 2|\Pi(G)| - 1$. □

This allows to bound the run-time of Algorithm 5.2 as follows:

Theorem 5.43. *Let G be a connected chordal graph. Then, Algorithm 5.2 runs in time at most $O(|\Pi(G)|^2 \cdot (n + m))$.*

Proof. By Proposition 5.42, `count` explores $\mathcal{O}(|\Pi(G)|)$ distinct recursive calls. For each of the corresponding chordal graphs, the clique tree is computed in time $\mathcal{O}(n + m)$. Afterwards, for each maximal clique, the subproblems are computed by Algorithm 5.1 in time $\mathcal{O}(n + m)$ by Theorem 5.14. The computation of $\phi_{T^R}(K)$ can be performed in time $O(m)$ as well by Corollary 5.38. □

It follows immediately by Proposition 5.2 and Theorem 5.34 and 5.43 that:

Theorem 1.4. *Let G be a CPDAG. There exists an algorithm computing the size of the Markov equivalence class $[G]$ in time $O(n^4)$.*

We note that the proof of Theorem 5.43 associates each distinct subproblem in the recursion with an S-flower. Moreover, the S-flowers can be connected to subtrees of the clique-tree of initial graph G (Lemma 5.23 item 2). It follows that the whole recursion could, in theory, be implemented by only relying on the initial clique-tree and passing subtrees of it to the recursive calls. As the clique-tree is a compact representation of a chordal graph, this could be the basis for further improvements of the asymptotic run-time. Exploring this is an interesting direction for future work.

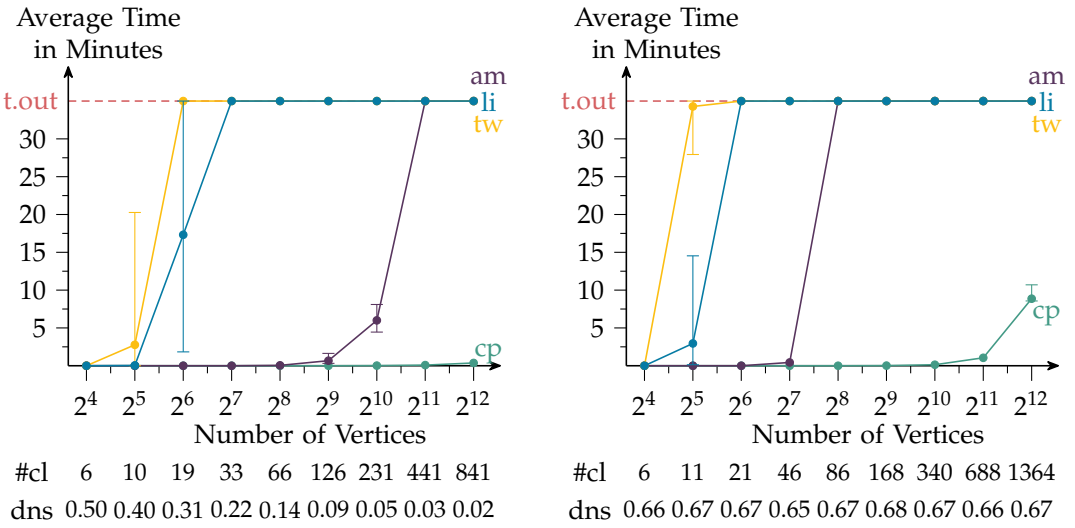


Figure 5.3: Experimental results for the solvers Clique-Picking (cp), AnonMAO (am), TreeMAO (tw), and LazyIter (li) on random chordal graphs with $n = 16, 32, \ldots, 4096$ vertices. For the left plot, we used graphs generated with the subtree intersection method and density parameter $k = \log n$; the right plot contains the results for random interval graphs. At the bottom, we present the number of maximal cliques as well as the graph density $|E|/\binom{|V|}{2}$.

5.3 Experimental Evaluation

We evaluate the practical performance of the Clique-Picking algorithm by comparing it to three state-of-the-art algorithms for #AMO. *AnonMAO* [Ganian et al., 2020] is the best root-picking method; *TreeMAO* [Talvitie and Koivisto, 2019] utilizes dynamic programming on the clique tree; and *LazyIter* [AhmadiTeshnizi et al., 2020] combines techniques from intervention design with dynamic programming.[4]

Figure 5.3 shows the run time of the four algorithms on random chordal graphs – details of the random graph generation and further experiments are given in Wienöbst et al. [2021b] and the source code is available at `github.com/mwien/CliquePicking`. We chose the random subtree intersection method (left plot in Figure 5.3) as it generates a broad range of chordal graphs [Seker et al., 2017]; and we complemented these with random interval graphs (right plot) as *AnonMAO* runs provably in polynomial time on this subclass of chordal graphs [Ganian et al., 2020].

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

[4]The implementation of LazyIter provided by the authors gives incorrect results for some instances. The authors have been informed in 2020, however, since then there has been no update.

5.4 Conclusions

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

6

Applications of Efficiently Counting Markov Equivalent DAGs

The previous section focused on the graph-theoretical and algorithmic contributions towards developing a polynomial-time algorithm for computing the size of an MEC represented by a CPDAG, which reduces to counting the AMOs of a chordal graph. This chapter considers the larger picture and, in particular, embeds the Clique-Picking algorithm into the broader context of causal discovery. We consider the following problems: Uniform sampling of DAGs from an MEC, choosing an intervention target in active learning of DAGs, computing the multiplicity of causal effects in an MEC and the general problem of counting consistent extensions of PDAGs. We discuss how they can be tackled using Clique-Picking directly, if possible, or by building on it.

These tasks have been previously connected to the problem of counting Markov equivalent DAGs, see in particular [Talvitie and Koivisto, 2019] and [Ghassami et al., 2019]. Our contributions lie in the following: Concerning the first application, i.e. uniform sampling, we perform the technical task of transforming the polynomial-time counting algorithm to a practical and worst-case polynomial-time uniform sampler. Particularly, we show its effectiveness by providing, to the best of our knowledge, the *first* implementation of an exact uniform sampling algorithm for Markov equivalent DAGs and analyzing it experimentally. The implementation is split into a preprocessing and a sampling phase such that the latter can be performed in linear-time per sampled DAG.

In the second and in particular the third application, we show that these problems can be analyzed under the framework of *interventional CPDAGs* (also known as *interventional essential graphs*). This allows us to improve the performance in various aspects, in particular, it is possible to use Clique-Picking (Algorithm 5.2) to again compute the number of consistent extensions (of the interventional CPDAG) in polynomial-time. Previously, e.g. as in Ghassami et al. [2019], these problems have been modeled with PDAGs without exploiting the structure present in interventional CPDAGs, thus leading to exponential time algorithms.

In the final section, we discuss this general problem of counting the number of consistent extensions of a PDAG in more detail. We show that this problem is #P-hard and thus likely not solvable in polynomial-time. We still sketch an algorithm based on Clique-Picking, which provides a reasonable line-of-attack for practical settings.

Generally in this chapter, we aim to show that the task of counting Markov equivalent DAGs does not have to be avoided in applications because it can be solved efficiently in theory and practice as our results from the previous section certify. This is in contrast to many common approaches, particularly for the second and third task, which prefer

other approaches, not relying on the counting task, sometimes at the cost of accuracy. In Sections 6.2 and 6.3 we go into more detail on this topic.

In contrast, it is *not* the goal of this chapter to show that these applications can *only* be practically tackled because of the polynomial-time Clique-Picking algorithm from the previous section. Quite oppositely, for many instances that are commonly encountered in causal discovery, which usually have rather small connected components (with regard to the undirected subgraph), more naive counting approaches can often be effective in practice as well.

6.1 Uniform Sampling of Markov Equivalent DAGs

The complexity of uniform sampling and counting are closely connected and their relation has been the topic of intensive study. In particular, it is well-known and easy to show that an efficient (polynomial-time) algorithm for counting configurations with self-reducible relations implies an efficient uniform sampling algorithm (see [Sinclair and Jerrum, 1989] and references therein). Self-reducible means that the original counting task can be formulated recursively, more precisely, that the number of configurations can be expressed by a combination of at most polynomially many solutions to smaller counting instances of the same problem. In the context of this work, the formula given in Proposition 5.32 is such a self-reduction thus making the problem at hand amenable to this technique.[1]

The main idea is to sample the configuration step-by-step, at each weighting the probability proportionally to the number of partial configurations for the remaining sequence. Concretely, for sampling of Markov equivalent DAGs, we aim to choose the first clique K in a topological ordering inducing an AMO proportionally to the number of AMOs counted at K in the Clique-Picking approach and draw a uniform permutation π_K of it that is in $\phi_{T^R}(K)$.

Algorithm 6.1: The recursive function `sample` uniformly samples an AMO (represented through its topological ordering) from a connected chordal graph G.

input : Connected chordal graph G.
output: Topological ordering of uniformly drawn AMO of G.

```
function sample(G)
    Let T^R be a rooted clique tree of G.
    K := drawn w. probability proportional to |φ_{T^R}(K)| × ∏_{H∈C(G^K)} |AMO(H)|
    τ := uniformly drawn from φ_{T^R}(K)
    foreach H ∈ C(G^K) do
        τ := τ + sample(H, memo)
    end
    return τ
end
```

[1]The first self-reduction for this problem was given by [He et al., 2015] as discussed in the previous chapter. Notably, this self-reduction could also be used in combination with Clique-Picking, by choosing the source vertex proportionally to the number of AMOs with this source (this number can be computed in polynomial-time using Clique-Picking as well) and this would also yield polynomial-time (as there are at most n steps of choosing a source and for each Clique-Picking is called at most n times). However, the run-time is rather high (of order n^6) and it is not clear how to use preprocessing to significantly improve it. In contrast, we will be able to achieve the same preprocessing run-time as for Clique-Picking, that is $O(n^4)$, and linear-time sampling afterwards.

This approach is formalized in Algorithm 6.1. The recursive function `sample` takes as input a connected chordal graph G and produces a topological ordering of the vertices τ, which represents a uniformly sampled AMO of G. It utilizes the formula

$$|\mathrm{AMO}(G)| = \sum_{K \in \Pi(G)} |\phi_{T^R}(K)| \times \prod_{H \in \mathcal{C}(G^K)} |\mathrm{AMO}(H)|$$

derived in Proposition 5.32. The idea is to first sample a maximal clique K, which is put at the start of the topological ordering. For this, K is drawn with probability proportional to

$$|\phi_{T^R}(K)| \times \prod_{H \in \mathcal{C}(G^K)} |\mathrm{AMO}(H)|,$$

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 $\phi_{T^R}(K)$. 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 6.1. *Given a connected chordal graph 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 clique to choose and, as $\phi_{T^R}(K)$ contains all permutations of K, such a permutation (and hence the corresponding AMO) is chosen uniformly.

In order to make the following arguments more precise, we denote with $\Pr(\tau_D(G))$ the probability that Algorithm 6.1 draws a topological ordering τ of the vertices in G that represents AMO D. Our goal is to show, as we just did in the base case, that for all D:

$$\Pr(\tau_D(G)) = 1/|\mathrm{AMO}(G)|.$$

For connected chordal graph G and clique-tree T^R, let K_D be the maximal clique, at which D is "counted" and let π_D be the corresponding permutation of K_D in any topological ordering of D. The correctness of the proof relies on the fact that both K_D and π_D are unique (for K_D this follows from the proof of Proposition 5.32). Then:

$$\begin{aligned}
\Pr(\tau_D(G)) &= \Pr(K_D)\Pr(\pi_D \mid K_D) \prod_{H \in \mathcal{C}(G^{K_D})} \Pr(\tau_{D[H]}(H)) \\
&= \frac{|\phi_{T^R}(K_D)| \times \prod_{H \in \mathcal{C}(G^{K_D})} |\mathrm{AMO}(H)|}{|\mathrm{AMO}(G)| \times |\phi_{T^R}(K_D)|} \prod_{H \in \mathcal{C}(G^K)} \Pr(\tau_{D[H]}(H)) \\
&= \frac{\prod_{H \in \mathcal{C}(G^{K_D})} |\mathrm{AMO}(H)|}{|\mathrm{AMO}(G)| \times \prod_{H \in \mathcal{C}(G^{K_D})} |\mathrm{AMO}(H)|} = \frac{1}{|\mathrm{AMO}(G)|}
\end{aligned}$$

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

$$\prod_{H \in \mathcal{C}(G^{K_D})} \Pr(\tau_{D[H]}(H)) = \frac{1}{\prod_{H \in \mathcal{C}(G^{K_D})} |\mathrm{AMO}(H)|}$$

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 6.1.

Note that when calling the function `sample` for an input graph G, it is only necessary to know for each maximal clique K, the following information: the set of forbidden prefixes used in the computation of $|\phi_{T^R}(K)|$ and

$$|\phi_{T^R}(K)| \times \prod_{H \in \mathcal{C}(G^K)} \#\mathrm{AMO}(H).$$

Moreover, in a single run of the counting algorithm (Algorithm 5.2) 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 maximal cliques K. 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 $\mathcal{O}(|K|^2)$ time. This leads to overall cost of $\mathcal{O}(n+m)$ of `sample`. Moreover, the precomputation is significantly more complicated, needing time $\mathcal{O}(|\Pi(G)|^2 \cdot n \cdot (n+m)$ and hence an additional factor n compared to the time complexity of Clique-Picking.

Here, we propose a simple Las Vegas algorithm for the implementation of line 4 based on rejection sampling. Due to the combinatorial structure of the counting function ϕ_{T^R}, 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., taking time $\mathcal{O}(|\Pi(G)|^2 \cdot (n+m))$. Afterwards, uniform sampling of an AMO is possible in linear-time $O(n+m)$, respectively $O(n)$ if only a topological ordering of the AMO is output.

Theorem 6.2. *There is an algorithm that, given a connected chordal graph G, uniformly samples an AMO of G in* expected time $\mathcal{O}(n+m)$*, respectively its topological ordering in $O(n)$, after an initial $\mathcal{O}(|\Pi(G)|^2 \cdot (n+m))$ setup.*

Proof. The main idea is simple: Implement line 4 in Algorithm 6.1 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 $|\phi_{T^R}(K)|$ be the number of allowed permutations and recall that, by Lemma 5.36, this coincides with $|\rho(X)|$ for an appropriately chosen sequence $X = (x_1, \ldots, x_k)$. The ratio $|\rho(X)|/x_k!$ gives the probability that a random permutation is allowed. We derive a lower bound of $1/2$ for the ratio in order to obtain the statement. The value of $|\rho(X)|$ reaches its minimum when allowing as few permutations as possible. Consequently, worst-case sequences X have the form $X = (1, 2, \ldots, x_k)$.

In this case, the number of allowed permutations is known as the number of irreducible permutations of $\{1, \ldots, x_k\}$ (OEIS A003319 [OEIS Foundation Inc., 2022]), which we denote with $|\rho(x_k)|$. It is well-known (and a special case of Lemma 5.36) that

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

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 $p \geq 8$

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

Hence,

$$\frac{|\rho(X)|}{x_k!} \geq \frac{1}{2}$$

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

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

The same bound carries over to $|\phi_{T^R}|$. It remains to analyze the expected run time of the overall routine. Drawing a permutation in $\phi_{T^R}(K)$ is possible in linear time in $|S|$. We do this by sampling a permutation of $\{1, \ldots, x_k\}$, which can easily be mapped to K (see proof of Lemma 5.36). Checking whether a permutation is forbidden can be done in linear-time as well: For every object $s \in \{1, \ldots, x_k\}$, we record its first "occurrence" in sequence X, that is the first i such that $x_i \geq s$, in $o(s)$. Afterwards, 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, 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 $\mathcal{O}(n)$ after appropriate preprocessing. Note that to output the AMO itself, $\Theta(n+m)$ time is needed as this is the size of the output. □

This immediately yields, similar to the case of Theorem 1.4.

Theorem 6.3. *Let G be a CPDAG. There exists an algorithm that uniformly samples a member of the Markov equivalence class $[G]$ in time $O(n+m)$ after an initial $O(n^4)$ setup.*

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 by 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 6.1: The run times in seconds of the standard Clique-Picking (CP) 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.[2]

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.

[2]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.

6.2 Interventional CPDAGs and Active Learning

As we discussed in detail 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 true underlying 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 CPDAG*. This graph (as the CPDAG for MECs) is the PDAG representation of the corresponding class of DAGs. An example and further explanation regarding interventional MECs and CPDAGs is given in Figure 6.1.

Let $D = (V, E)$ be a DAG and joint probability distribution $\mathbb{P}(x_1, \ldots, x_n)$ be Markovian to D. For a set of targets $I \subseteq V$, an intervention with perturbation targets $v \in I$ models the effect of replacing the observational distribution $\mathbb{P}(x_v \mid x_{Pa(v)})$ by $\mathbb{P}^I(x_v)$ for all $v \in I$. 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 $(\mathbb{P}, \{\mathbb{P}^I\}_{I\in\mathcal{I}})$ is $\mathcal{I}$-Markovian to D if $\mathbb{P}$ is Markovian to D and for all $I \in \mathcal{I}$ the interventional distribution $\mathbb{P}^I$ factors as

$$\mathbb{P}^I(x_1, \ldots, x_n) = \prod_{v\notin I} \mathbb{P}(x_v \mid x_{Pa(v)}) \prod_{v\in I} \mathbb{P}^I(x_v).$$

Two DAGs D_1 and D_2 are $\mathcal{I}$-Markov equivalent if for all positive distributions, $(\mathbb{P}, \{\mathbb{P}^I\}_{I\in\mathcal{I}})$ is $\mathcal{I}$-Markovian to D_1 if and only if $(\mathbb{P}, \{\mathbb{P}^I\}_{I\in\mathcal{I}})$ is $\mathcal{I}$-Markovian to D_2. This relation can be expressed in graphical language as follows: For a conservative family of targets[3] $\mathcal{I}$, 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 $\mathcal{I}$-CPDAG $\mathcal{E}_\mathcal{I}(D) = \text{pdag}([D]_\mathcal{I})$. A graph G is called an $\mathcal{I}$-CPDAG if $G = \mathcal{E}_\mathcal{I}(D)$ for some DAG D.

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

Proposition 6.4 (Hauser and Bühlmann [2012])**.** *Let G be an $\mathcal{I}$-CPDAG representing an $\mathcal{I}$-MEC $[D]_\mathcal{I}$ for a target family $\mathcal{I}$. Then, the connected components $\mathcal{C}(G)$ are chordal. 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 connected components of G independently of each other.*

For example, in the $\mathcal{I}$-CPDAG G_4 in Figure 6.1 representing the $\mathcal{I}$-MEC determined by the intervention result $c \to a \leftarrow b$, the only non-trivial connected component is the triangle $b \frown c \frown d$. The $\mathcal{I}$-MEC consists all DAGs which can be obtained from G by acyclic moral orientations of the triangle. Generally, the proposition above implies that for interventional CPDAG G, the number of DAGs in the corresponding equivalence class $[D]_\mathcal{I}$ is

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

[3] 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.

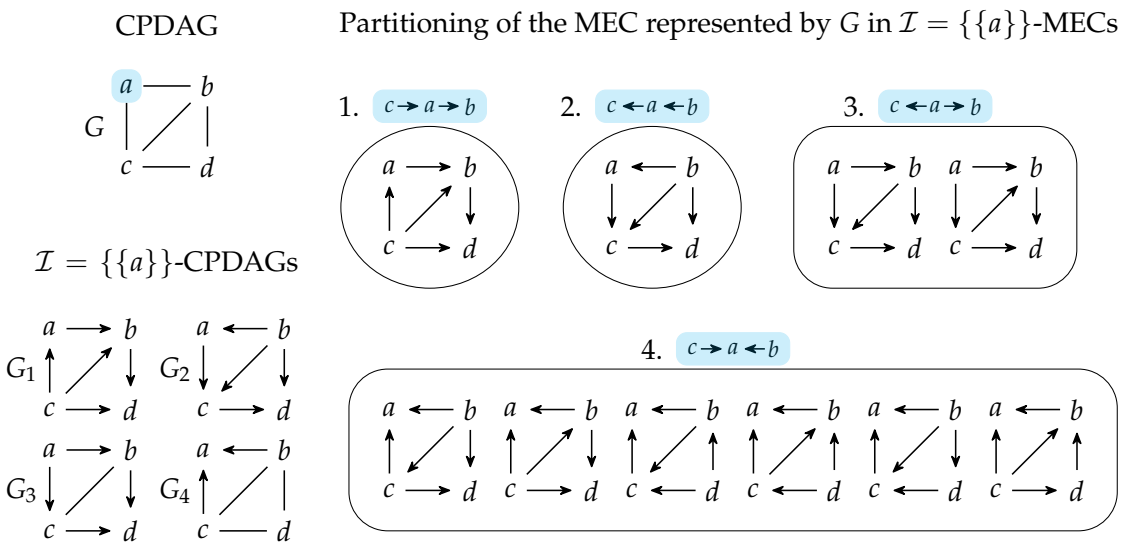


Figure 6.1: For the CPDAG G on the top left, we show on the right the interventional MECs for the family $\mathcal{I} = \{\{a\}\}$, i.e., an intervention is performed on the variable corresponding to vertex a (marked in color). The possible results for the intervention are shown in the colored regions and the DAGs are partitioned according to those configurations. Each $\{\{a\}\}$-MEC can be represented by the corresponding interventional CPDAG on the bottom left, which encodes the still unknown edge orientations as undirected edges.

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 1.4 and 6.3.

Theorem 6.5. *For a given interventional CPDAG 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 CPDAGs can be computed efficiently can be utilized, e.g., 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 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 Figure 6.1 for an illustration). Additionally, we assume variables are manipulated sequentially, i.e., one can use intervention results obtained by ma-

nipulating 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 CPDAG 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, effective counting algorithms are central 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 $[D]_{\mathcal{I}}$, for $\mathcal{I} = \{\emptyset\}$ and D being the true DAG, which is represented as an $\mathcal{I}$-CPDAG $\mathcal{E}_{\mathcal{I}}(D)$. 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

$$v^* = \arg\min_{v \in V} \max_{D' \in [D]_{\mathcal{I}}} |[D']_{\mathcal{I} \cup \{\{v\}\}}|. \tag{6.1}$$

MaxEntropy chooses

$$v^* = \arg\max_{v \in V} H_v, \tag{6.2}$$

where H_v is the entropy defined as follows: Let $D_1, \ldots, D_k \in [D]_{\mathcal{I}}$ be DAGs such that $[D_1]_{\mathcal{I} \cup \{\{v\}\}} \dot\cup \ldots \dot\cup [D_k]_{\mathcal{I} \cup \{\{v\}\}}$ 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

$$v^* = \arg\min_{v \in V} \max_{D' \in [D]_{\mathcal{I}}} \xi(\mathcal{E}_{\mathcal{I} \cup \{\{v\}\}}(D')), \tag{6.3}$$

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 greedy heuristics.

Example 6.6. *For the CPDAG G in Figure 6.1, 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 $v = a$ is shown in Figure 6.1. The values needed to select $v^* = b$ or $v^* = c$ solving the Equation* (6.1)*, Equation* (6.2)*, resp. Equation* (6.3)*, are given in the table below.*

v	cardinalities of $\{\{v\}\}$-CPDAGs	max card.	H_v	number of undir. edges in $\{\{v\}\}$-CPDAGs	max num.
a	6, 2, 1, 1	6	1.57	3, 1, 0, 0	3
b	3, 2, 2, 1, 1, 1	3	2.45	2, 1, 1, 0, 0, 0	2
c	3, 2, 2, 1, 1, 1	3	2.45	2, 1, 1, 0, 0, 0	2
d	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 6.5, the sizes of MECs needed to choose a vertex with regard to Equation (6.1), respectively Equation (6.2), can be computed in polynomial time, assuming the $(\mathcal{I} \cup \{\{v\}\})$-MECs are represented as interventional CPDAGs.

Another efficiency issue of the algorithms, including OptSingle, concerns the computation of the interventional CPDAGs 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). Using the ideas from the previous chapter, especially from Algorithm 5.1, we can show that, given a current $\mathcal{I}$-interventional CPDAG G and an interventional result on a vertex v, we can compute the new interventional CPDAG in linear time. This is possible using an algorithm based on MCS [Tarjan and Yannakakis, 1984] similar to Algorithm 5.1, which we describe 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 connected component of G containing v. Then the resulting orientations of the incident undirected edges $u - v$ in G can be represented as a clique $K \subseteq Ne_H(v)$, which contains the incident vertices u of edges oriented as $u \to v$;[4] the edges $u - v$, with $u \in Ne_H(v) \setminus K$, are oriented as $u \leftarrow v$. Note, that K can be empty. E.g. in [Schauer and Wienöbst, 2023], it is discussed how to elegantly enumerate all cliques in a chordal graph (the subgraph induced on the neighbors of vertex v).

Theorem 6.7. *Assume D is a DAG, $\mathcal{I}$ is a target family, v is a vertex, and H is a connected component of $G = \mathcal{E}_\mathcal{I}(D)$ containing v. Let a clique $K \subseteq Ne_H(v)$ 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 interventional CPDAG $G' = \mathcal{E}_{\mathcal{I}\cup\{\{v\}\}}(D_K)$ can be computed in time $\mathcal{O}(n + m)$.*

Proof. Due to Proposition 6.4 we know that, to compute G', it is sufficient to maximally orient H into H' since the remaining connected components of G remain unchanged. Let X be the set of vertices reachable from v (including v itself) in H with edges incident to K removed. Let $A = V \setminus \{X \cup K\}$ 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 X, (iii) the edges from K to X are oriented outwards from K (iv) and the edges in $H[X]$ are given by calling Algorithm 5.1 on $H[X \cup K]$ 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 $H[X]$ 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 $H'[X]$. Consequently, every edge between K and X has to be oriented from K to X 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 X is oriented from $A \cup K$ to X. It follows that the chordal induced subgraph $H[A \cup K]$ can be oriented independently of the remaining graph as no v-structure or 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 5.1 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 5.14 it is argued that there exists an AMO with $a \to b$ and one with $a \leftarrow b$. Now note that finding an AMO for $H[X \cup$

[4] 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.

$K]$ (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. □

This theorem improves upon previous work by AhmadiTeshnizi et al. [2020], which gave an $O(d \cdot m)$ algorithm for this task.

For an example of our whole approach, computing a vertex solving Equation (6.1) in MinMaxMEC can be implemented as shown in Algorithm 6.2 below. The other two approaches can be handled similarly. The time complexity is $\mathcal{O}(|\Pi(v)| \cdot n^4)$, where $|\Pi(v)|$ is the number of possible intervention results (coinciding with the number of cliques in the neighborhood of v) and n^4 is the time complexity of the Clique-Picking algorithm.

Algorithm 6.2: An efficient implementation of MinMaxMEC: The algorithm computes a vertex solving Equation (6.1).

input : An $\mathcal{I}$-CPDAG $G = (V, E, A)$.
output: Vertex v^* solving Equation (6.1).

if *G is a DAG* **then return** $\emptyset$
minmax $:= \infty$
foreach *undirected component H of G* **do**
  foreach $v \in H$ **do**
    max $:= 0$
    foreach *clique* $K \subseteq Ne_H(v)$ **do**
      $G' := \mathcal{E}_{\mathcal{I} \cup \{\{v\}\}}(D^K)$ `// compute with Theorem 6.7`
      $c := |\text{EXT}(G')|$ `// compute with Algorithm 5.2`
      if $c >$ *Max* **then** Max $:= c$
    end
    if *Max* $<$ *MinMax* **then** MinMax $:=$ Max; $v^* := v$
  end
end
return v^*

These improvements can make the difference between infeasibility and practical applicability. We replicate the 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 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[5] of our methods, these approaches scale to much larger graphs and that they deliver superior results compared to other algorithms.

In Figure 6.2, 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 need the least amount of interventions. In case of the sparse graphs the differences are larger due to the fact that the structure

[5]The experiments in this paper record the results using implementations in the Julia programming language provided under `https://github.com/mwien/counting-with-applications`.

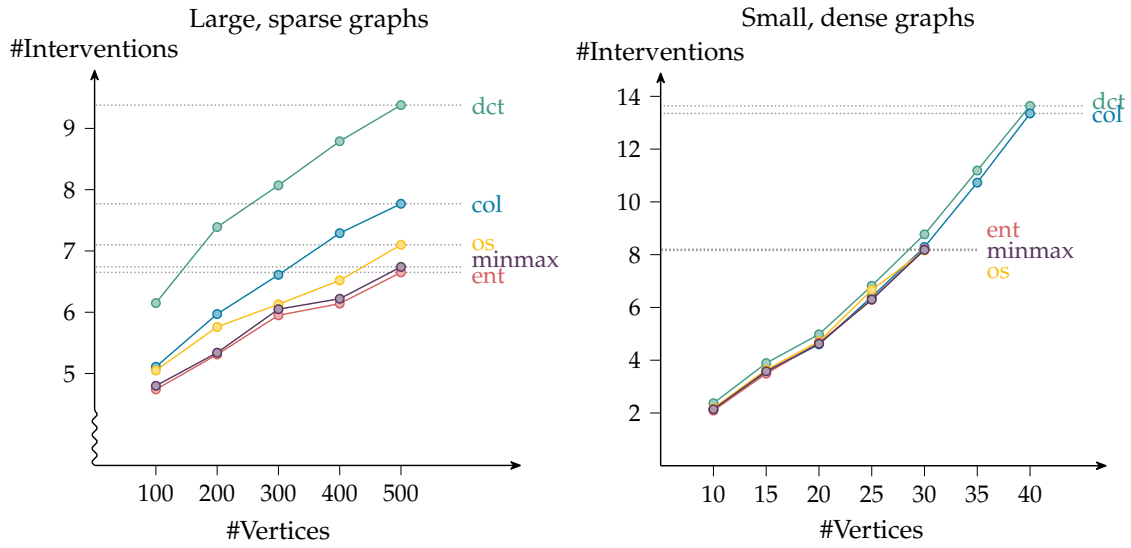


Figure 6.2: We compare the performance of the three discussed strategies MinMaxMEC (minmax), MaxEntropy (ent) and OptSingle (os), with the coloring-based strategy (col) of Shanmugam et al. [2015], and directed-clique-tree approach (dct) of Squires et al. [2020]. On the left, the average number of interventions 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).

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[6]). In case of the dense graphs, the implementations of MinMaxMEC, MaxEntropy and OptSingle can handle up 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.

6.3 Estimating Causal Effects from CPDAGs and Observed Data

A second application concerns calculating *total causal effects* to measure the effects of interventions based on 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

[6]The experiments were run on an Intel(R) Core(TM) i7-8565U CPU with 16 GBs of RAM.

X_i, for $i \in V$, and a distribution $\mathbb{P}$ over $X = (X_1, \ldots, X_n)$, which is Markovian to D, the distribution generated by an intervention on X_i, written $do(X_i = x_i')$, can be expressed in the truncated factorization formula

$$\mathbb{P}(x_1, \ldots, x_n \mid do(X_i = x_i')) = \prod_{j=1, j\neq i}^{n} \mathbb{P}(x_j \mid x_{Pa(j)})$$

if $x_i = x_i'$ and 0 otherwise. By marginalizing 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):

$$\mathbb{P}(x_j \mid do(X_i = x_i')) \;=\; \begin{cases} \mathbb{P}(x_j) & \text{if } j \in Pa(i) \\ \sum_{x_{Pa(i)}} \mathbb{P}(x_j \mid x_i', x_{Pa(i)}) \mathbb{P}(x_{Pa(i)}) & \text{if } j \notin Pa(i). \end{cases} \tag{6.4}$$

In turn, the *total causal effect* θ_{ji} of X_i on X_j is defined as a quantity of the interventional distribution with a common choice being, in the non-parametric setting, the difference of expected values under two actions x_i' and x_i'' [Pearl, 2009, page 70]:

$$\theta_{ji}(D) \;=\; \mathbb{E}(X_j \mid do(X_i = x_i')) - \mathbb{E}(X_j \mid do(X_i = x_i'')). \tag{6.5}$$

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 causal effects when the formula (6.4) is not applicable, e.g., due to unobserved 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 $D_1, \ldots, D_k \in [G]$ be DAGs such that $[D_1]_{\{\emptyset,\{i\}\}} \dot\cup \ldots \dot\cup [D_k]_{\{\emptyset,\{i\}\}}$ is the partition of $[G]$ into the possible interventional-MECs for $\mathcal{I} = \{\emptyset, \{i\}\}$. Then, for all $\ell = 1, \ldots, k$, 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.[7]

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 $\Theta^L = \{\theta_{ji}(D_1), \ldots, \theta_{ji}(D_k)\}$ 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.

[7] 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 6.5 and connects this and the previous application discussed in this chapter.

In the context of Figure 6.1, 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 6.3. Relying on the formulation of the problem in terms of $\mathcal{I}$-MECs from above and utilizing Theorem 6.5 and 6.7, we can conclude:

Theorem 6.8. *For a given CPDAG $G = (V, E)$ and $i, j \in V$, Algorithm 6.3 computes the multiset $\Theta = \{\theta_{ji}(D) \mid D \in [G]\}$ of causal effects of X_i on X_j for all DAGs $D \in [G]$. It runs in time $\mathcal{O}(k \cdot n^4)$, where $k = |\Theta^L|$ (disregarding the time to estimate the causal effect $\theta_{ji}(D)$ from data, respectively assuming it can be computed in $O(1)$ for a DAG D).*

Algorithm 6.3: An efficient implementation of the global-IDA algorithm.

input : A CPDAG $G = (V, E, A)$ and vertices $i, j \in V$.
output: Multiset Θ of possible causal effects of X_i on X_j

$\Theta := \emptyset$
$H :=$ connected component of G containing vertex i
foreach *clique* $K \subseteq Ne_H(i)$ **do**
  $G' := \mathcal{E}_{\{\emptyset,\{i\}\}}(D_K)$ `// compute with Theorem 6.7`
  $c := |\mathrm{EXT}(G')|$ `// compute with Algorithm 5.2`
  Add c copies of $\theta_{ji}(D_K)$ to Θ.
end
return Θ

Hence, the additional effort of Algorithm 6.3 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].

In a causal *linear model*, where every edge represents a linear direct causal effect and under the assumption that the distribution of variables follows a multivariate normal distribution, one can compute the causal effects $\theta_{ji}(D)$, 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., [Pearl et al., 2016]). 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 $X_1, \ldots, X_p, X_{p+1}$ 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 $Y = X_{p+1}$.

They report, that in simulation studies over sparse DAGs, already for $p = 14$, at least one of the 10 replicates[8] of the global-IDA algorithm took more than 48 hours to compute, so that the computation was aborted. Correspondingly, for the *riboflavin data* with $p = 4088$ 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

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

specified parents, as proposed in Algorithm 6.3. Hence, local-IDA would be identical to performing step (i), whereas global-IDA would consist of (i) and (ii). Table 6.2 shows the results.

Table 6.2: Mean run-time 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 $p = 4088$ 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

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.

6.4 Counting Consistent Extensions of PDAGs

In the other settings in this chapter, it was possible to immediately build on the Clique-Picking algorithm using it as a preprocessing step, such as for uniform sampling, or to efficiently count the number of consistent extensions, for example in the case of an interventional CPDAG. In this section, we consider the more general case of counting the number of consistent extensions of a PDAG, which is more complex.

The following theorem shows that this problem is not in P under standard complexity-theoretic assumptions. We proceed by reducing 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 6.9. *The problem #EXT is #P-complete for arbitrary input graphs G, and in particular for PDAGs and MPDAGs.*

Proof. We give a parsimonious reduction from #TO to #EXT for PDAGs. The resulting PDAG can in turn be transformed into an equivalent MPDAG in polynomial time [Meek, 1995], which certifies hardness for this model class as well.

Given an instance for #TO, that is 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 directly solve these counting problems can be 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 of connected chordal graphs, 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 6.10 (ϕ'_{T^R})**.** *Let G be a connected chordal graph, $K \in \Pi(G)$ and T^R a rooted clique-tree. For a partial order $\preceq$ over the elements of K, we define $\phi'_{T^R}(K, \preceq)$ as the set of all permutations of K consistent with $\preceq$ that do not have a set $K' \in S$ as prefix, with $S = K_i \cap K_j$ for $K_i - K_j$ on the path from R to K in T^R.*

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. For example for $K = R$, it coincides with the problem of counting the extensions of a partial order. As this is equivalent to the problem #TO (all relations can be encoded as directed edges), which will be fundamental to our approach, we will denote by $\mathrm{TO}(K, \preceq)$ the set of linear orderings of K consistent with $\preceq$.

Lemma 6.11. *Let G be a connected chordal graph, $K \in \Pi(G)$, T^R a rooted clique-tree and $\preceq$ a partial order over the elements of K. Then, function $\phi'_{T^R}(K, \preceq)$ can be computed by $\mathcal{O}(\ell^2)$ calls to #TO.*

Proof. Note that $\phi'_{T^R}(K, \preceq)$ can be mapped to an appropriately defined function $\rho'(X, \preceq')$ with modified $\preceq'$ in analogy to Lemma 5.36. Thus, we base our approach on the recursive formula derived in Theorem 5.37:

$$|\rho(X)| = x_k! - \sum_{i=1}^{k} (x_k - x_i)! \cdot |\rho((x_1, \ldots, x_i))|.$$

Instead of $x_k!$, compute the number of permutations of $\{1, \ldots, x_k\}$ consistent with $\preceq'$. In the sum, check whether the partition in $\{1, \ldots, x_i\}$ (at the beginning of the permutation) and $\{x_i + 1, \ldots, x_k\}$ (at the end of the permutation) violates the partial order (let $I(x_i, \preceq')$

denote this and evaluate to 0 if $\preceq$ is violated, else to 1). Replace $\{x_i + 1, \ldots, x_k\}$ by the number of permutations of this subset of K which conforms to $\preceq$. We obtain:

$$|\rho'(X, \preceq')| = |\mathrm{TO}(\{1, \ldots, x_k\}, \preceq')| - \sum_{i=1}^{k} I(x_i, \preceq') \cdot |\mathrm{TO}(\{x_i + 1, \ldots, x_k\})| \cdot |\rho'((x_1, \ldots, x_i), \preceq')|.$$

Correctness follows as in Theorem 5.37 and as there are at most k recursive calls, we have $\mathcal{O}(k^2)$ calls to #TO. □

Theorem 6.12. *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 (see the technique used in the proof of Theorem 3.30). Note that a PDAG or an MPDAG represents a subset of a Markov equivalence class, C is the CPDAG of the full class.

2. Consider the connected components 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 connected component by calling a modified version of `count` from Algorithm 5.2 with additional parameter $\preceq$ and ϕ replaced by ϕ'. We pass this function a connected component of C and as $\preceq$ we choose $\preceq_G$, i.e., the partial ordering over the vertices of the connected component given by the directed edges of G (i.e., $u \preceq_G v$ if $u \to v$ in G or $u = v$).

The correctness follows immediately, as Algorithm 5.2 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 $\mathcal{O}(n^4)$. □

In practice, the bound of $\mathcal{O}(n^4)$ oracle calls should be rather pessimistic as the parameter k in the computation ϕ' is usually extremely small, at least for random chordal graphs, as we observed during experiments.

Sharma [2023] independently investigated the generalization of Clique-Picking to PDAGs, taking a parameterized perspective [Downey and Fellows, 2012] and deriving an $O(n^4 \cdot k! k^2)$ algorithm with k being the *clique-knowledge* (different from the k above), defined as the maximum number of vertices in a clique in the input graph that are incident to a "background edge" lying fully in the clique.

6.5 Conclusions

We investigated the practical aspects and applications of a polynomial-time algorithm for counting Markov equivalent DAGs. For the uniform sampling problem, we gave a novel 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. Our implementation is the first for this task. We also considered two applications

in the context of computing the size of subclasses of MECs, both given by an interventional CPDAG, whose size can also be efficiently computed. This means that in these applications the counting task does *not* have to be avoided, which enables researchers to choose more reliable and robust algorithms. Finally, we investigated the complexity of the general problem of counting the number of consistent extensions of PDAGs. One the one hand, we showed the #P-hardness, while, on the other, we provide an algorithmic approach based on Clique-Picking, connecting it to the problem of counting topological orderings.

7

Markov Equivalence of MAGs

DAGs lie at the center of the graphical formalization of causality. However, they bring a number of implicit assumptions about the underlying system of interest. Clearly, it is necessary to assume that there are no feedback loops, i.e. no directed cycles – an assumption which is often violated in practice [Mooij and Heskes, 2013]. But even if there exists an underlying DAG representation, in most cases not every variable is observed. Particularly, if there are unobserved common causes, it is easy to see that it is not possible to represent the causal relationships by a DAG over only the *observed* variables. It follows that in causal discovery *without* the assumption of unobserved common causes, more expressive models need to be considered.

A fundamental model class in this context are *maximal ancestral graphs*, MAGs for short [Richardson and Spirtes, 2002]. They are able to represent systems with unobserved confounders as well as selection bias. A variable is latent or unobserved if it is not measured or recorded. For example, the DAG G_1 in Figure 7.1 shows a causal structure over four observed variables represented by vertices a, b, c, d and a latent variable represented by u. DAG G_1 implies the independence relations $X_a \perp\!\!\!\perp \{X_c, X_d\}$ and $X_c \perp\!\!\!\perp \{X_a, X_b\}$ over the observed variables, i.e., after marginalizing variable X_u out. However, there is no DAG representing precisely these conditional independencies (CIs), which shows that DAGs are not closed under marginalization. One can represent the conditional independencies using MAGs as shown by graph G_2 in Figure 7.1. Additionally, DAGs are not expressive enough to handle selection variables, which are unmeasured variables determining whether a measured unit is included in the data, and are hence also not closed under conditioning. In contrast, the class of independence models associated with ancestral graphs is closed under marginalization and conditioning and the smallest such class which contains the DAG independence models [Richardson and Spirtes, 2002].

Despite many advances, a number of fundamental problems concerning the properties and algorithmic aspects of this important model class remain to be explored. We investigate the Markov equivalence of MAGs – one of the basic problems in this field. As for DAGs, MAGs that encode the same conditional independencies are said to be

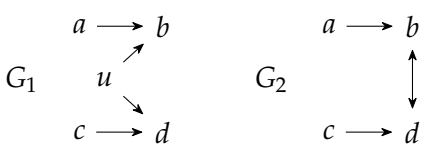


Figure 7.1: DAG G_1 encodes the CIs $X_a \perp\!\!\!\perp \{X_c, X_d\}$ and $X_c \perp\!\!\!\perp \{X_a, X_b\}$ among observed variables (u is a latent variable). G_2 is a MAG which encodes the same CIs. The example is from [Richardson and Spirtes, 2002, Figure 10].

Markov equivalent. In graphical language, we express conditional independencies via *m-separations*, an extended form of *d*-separation in DAGs (formal definitions are provided in Section 7.1).

An effective polynomial-time algorithm to test whether two MAGs are Markov equivalent has been the subject of significant research. A naive implementation of the definition requires testing *m*-separation relations over all pairs of vertices and all subsets of vertices, which takes exponential time. The first graphical criterion was given by Spirtes and Richardson [1997]: The *Spirtes and Richardson Criterion (SRC)* extends the conditions by Verma and Pearl [1990] and Frydenberg [1990] for DAGs and is based on the concept of *discriminating paths*. The SRC is intuitive and forms the basis of subsequent work. However, testing the SRC naively requires exponential time since there can be exponentially many discriminating path, which all have to be inspected. Zhao et al. [2005] proposed another, very elegant characterization using the concept of minimal collider paths, which however did not lead to polynomial time either. The first criterion that can be checked in polynomial time has been proposed by Ali et al. [2009]. The complexity of their method is bounded by $O(n \cdot m^4)$ for MAGs with n vertices and m edges.[1] Recently, a criterion based on parametrizing sets was proposed by Hu and Evans [2020]. These sets can be generated in time $O(n \cdot m^2)$ (for dense graphs with $m \in \Omega(n^2)$ this equates to $O(n^5)$) leading to a faster algorithm.

The main contribution of this paper is a new criterion for the Markov equivalence of MAGs. It is a simple and constructive variant of the SRC and allows us to develop an algorithm for equivalence testing in *cubic time*. This breaks the previous $O(n^5)$ worst-case time barrier. Our criterion, coined *constructive-SRC*, is based on discriminating paths, but it avoids searching through exponentially many paths and boils down to a simple graphical condition. The constructive-SRC is intuitive and checking it by hand is convenient. For sparse graphs with maximal degree Δ, which are common in causal modeling, the running time is bounded by $O(n \cdot \Delta^2)$. We compare our algorithm experimentally with the algorithm by Hu and Evans [2020] and show that the theoretical improvements lead to a better practical performance.

Obtaining the cubic run-time raises the question of whether further improvements are possible, e.g., whether a run-time of $O(n^2)$ can be attained. We discuss this issue by relating it to the Markov equivalence of DAGs, where such a run-time is achievable using the CPDAG representation of Markov equivalence classes. We uncover obstacles in translating this approach towards the MAG setting, while also highlighting related open research questions in this area.

7.1 Background

In this chapter, we consider graphs $G = (V, E, A, B)$ containing undirected $a - b$, directed $a \to b$ and bidirected $a \leftrightarrow b$ edges. In addition to the notion of ancestors and descendants introduced in Chapter 2, $Dis_G(v)$ denotes the set of vertices in the same district as v, i.e., the ones connected to v via bidirected edges. When we speak of a *v-structure* in this chapter, also called an *unshielded collider*, we mean an ordered triple of vertices (u, c, v) that induces the subgraph $u \ast\!\!\to c \leftarrow\!\!\ast v$. The $\ast$ indicates that any edge mark is possible. As before, vertex c on a path π is called a *collider* if two arrowheads of π meet at c, e.g. if π contains $u \leftrightarrow c \leftarrow v$. Two vertices are *collider connected* if there is a path (a *collider path*) between them on which all internal vertices are colliders; hence, adjacent vertices

[1] Concurrently to our work, Claassen and Bucur [2022] showed that based on the results by Ali et al. [2009] an $O(n^4)$ algorithm can be derived.

Figure 7.2: A discriminating path from x to y for b. For the last three vertices, $\leftrightarrow b \leftrightarrow y$, $\leftrightarrow b \rightarrow y$, and $\leftarrow b \rightarrow y$ are possible configurations (see Lemma 7.7). In the first one b is a collider, in the other two a non-collider.

are collider connected. Vertices are *m-connected* by a set Z if there is a path π between them on which every collider is in $An(Z)$ and every vertex that is not a collider is not in Z. Such a π is called an *m-connecting path given* Z. If vertices u, v are not m-connected by Z, written as $(u \perp\!\!\!\perp v \mid Z)_G$, we say that Z *m-separates* them. Two sets X, Y are m-separated by Z if all their vertices are pairwise m-separated by Z. In DAGs, m-separation is equivalent to d-separation [Pearl, 2009].

A graph $G = (V, E, A, B)$ is called *ancestral* (AG) if *(i)* it is acyclic, *(ii)* for every bidirected edge $a \leftrightarrow b$ vertex a is not an ancestor of b (and vice versa), and *(iii)* for every undirected edge $a - b$ vertex a (and vertex b) have no parents or siblings. Consequently, ancestral graphs contain at most one edge type between two vertices. An AG is a *maximal ancestral graph* (MAG) if set Z exists for every pair of nonadjacent vertices a and b such that a and b are m-separated by Z. Every AG can be turned into an equivalent MAG by adding bidirected edges between vertices that cannot be m-separated. Syntactically, all DAGs are MAGs and all AGs that contain only directed edges are DAGs.

Two AGs G_1 and G_2 with the same vertex set V are said to be *Markov equivalent* if we have for all pairwise disjoint sets $A, B, Z \subseteq V$ with $A \neq \emptyset$ and $B \neq \emptyset$ that A and B are m-separated given Z in G_1 if, and only if, A and B are m-separated given Z in G_2. The following definition is central for the study of Markov equivalence of MAGs:

Definition 7.1 (Discriminating path [Richardson and Spirtes, 2002])**.** *In a MAG G, a path $\pi = (x, q_1, \ldots, q_p, b, y)$, $p \geq 1$, is called* discriminating *for vertex b if*

(i) x is not adjacent to y and

(ii) any q_i, for $1 \leq i \leq p$, is a collider on π and a parent of y.

A discriminating path is illustrated in Figure 7.2. For vertices b and y in G denote by $Discr_G(b, y)$ the set of all discriminating paths $\pi = (x, q_1, \ldots, q_p, b, y)$ for b. Our focus lies on the computational complexity of the following problem:[2]

Problem 7.2. MAG-EQUIVALENCE
Instance: *Two MAGs G_1 and G_2.*
Question: *Are G_1 and G_2 Markov equivalent?*

7.2 Previous Work

A graphical criterion for Markov equivalence of DAGs was provided by Verma and Pearl [1990] and Frydenberg [1990], which we stated in Theorem 2.4. It stated that two DAGs are Markov equivalent iff they have the same adjacencies and unshielded colliders. The first graphical criterion for two *MAGs* to be Markov equivalent was given by Spirtes and Richardson [1997]:

[2]We first deal with the problem for MAGs *without undirected edges*. We later discuss in Section 7.6 how these can be included with minor modifications (our main theorem holds as is).

Theorem 7.3 (Spirtes and Richardson Criterion (SRC)). *Two MAGs G_1 and G_2 are Markov equivalent if, and only if,*

(i) *G_1 and G_2 have the same adjacencies,*

(ii) *G_1 and G_2 have the same unshielded colliders, and*

(iii) *if π forms a discriminating path for b in G_1 and G_2, then b is a collider on the path π in G_1 if, and only if, it is a collider on the path π in G_2.*

Note that it is indeed possible that G_1 contains a discriminating path for b and y, which is not present in G_2, even in the case of Markov equivalence (see examples 2 and 3 in Figure 7.3). Therefore, testing property *(iii)* naively requires exponential time as one has to consider all discriminating paths for variable b (which may be exponentially many).[3]

Later, Zhao et al. [2005] proposed the following elegant characterization:

Theorem 7.4 (Zhao et al. [2005]). *Two MAGs G_1 and G_2 are Markov equivalent if, and only, if G_1 and G_2 have the same minimal collider paths.*[4]

However, this result also does not lead to a polynomial-time algorithm as there can be exponentially many minimal collider paths. Subsequently, discernible effort has been made to develop an algorithm that tests whether two MAGs are Markov equivalent and that runs in polynomial time [Ali et al., 2009, Hu and Evans, 2020]. To achieve this, the natural formulation in the style of Theorem 7.3 has been abandoned and more involved criteria without an intuitive graphical interpretation were introduced.

Ali et al. [2009] used *triples with order* (if the triple forms a collider, it is called a *collider with order*). The idea behind this approach is to consider only the discriminating paths that are present in any Markov equivalent MAG. The result is a crucial contribution towards characterizing Markov equivalence classes of MAGs [Ali et al., 2005], in particular for the completeness of the FCI algorithm [Zhang, 2008b], however the recursive definition of such triples lacks the graphical intuitiveness of, e. g, the SRC. With significant technical effort, the following criterion was developed:

Theorem 7.5 (Theorem 3.7 in [Ali et al., 2009]). *Two MAGs G_1 and G_2 are Markov equivalent if, and only if, they have the same adjacencies and the same colliders with order.*

This criterion led to the sought polynomial-time algorithm. However, the dependency is stated as $O(n \cdot m^4)$ for MAGs with n vertices and m edges. Later, and concurrently to our work, Claassen and Bucur [2022] gave an implementation of this approach running in time $O(n^4)$.

Another criterion was proposed by Hu and Evans [2020] based on so-called *parametrizing sets*. As we compare our algorithm with this approach, we give a brief overview. For a vertex set $W \subseteq V$, the *barren subset* of W is defined as $\text{barren}(W) = \{w \in W \mid De(w) \cap W = \{w\}\}$. A set H is called a *head* if $\text{barren}(H) = H$ and H is contained in a single district in $G[An(H)]$. Let $\mathcal{H}(G)$ be the set of heads and define the *tail* of a head as:

$$\text{tail}(H) = (Dis_{G[An(H)]}(H) \setminus H) \cup Pa_G(Dis_{G[An(H)]}(H)).$$

[3] Spirtes and Richardson [1997] claimed that the criterion is testable in polynomial time, which was later withdrawn [Ali et al., 2009].

[4] $\pi = (v_1, \ldots, v_p)$ is minimal if there is no order preserving subsequence $(v_1 = v_{i_1}, \ldots, v_t = v_{i_t})$ that forms a collider path.

The parametrizing set of MAG G is defined as the set $\mathcal{S}(G) = \{H \cup A \mid H \in \mathcal{H}(G) \text{ and } A \subseteq \text{tail}(H)\}$. Hu and Evans [2020] showed that MAGs G_1 and G_2 are Markov equivalent if, and only if, they have the same parametrizing sets. However, generating these sets is costly as they may have exponential size. Hence, they consider $\tilde{\mathcal{S}}_3 \subseteq \mathcal{S}$, which only includes sets S of cardinality 2 and 3, with the vertices in S having 1 or 2 adjacencies.

Theorem 7.6 (Corollary 3.2.1 in Hu and Evans [2020])**.** *Two MAGs G_1 and G_2 are Markov equivalent if, and only if, $\tilde{\mathcal{S}}_3(G_1) = \tilde{\mathcal{S}}_3(G_2)$.*

The sets $\tilde{\mathcal{S}}_3(G)$ can be generated in time $O(n \cdot m^2)$, which is significantly faster than the algorithm by Ali et al. [2009]. However, the criterion in this form is quite technical and does not lend itself easily to graphical characterizations of Markov equivalent MAGs.

7.3 A New Constructive Criterion for Markov Equivalence of MAGs

We propose a *constructive* variant of the Spirtes and Richardson Criterion (SRC) for the Markov equivalence of MAGs. This allows us to develop an efficient equivalence test, improving upon the previous $O(n^5)$ run-time by Hu and Evans [2020]. Additionally, our criterion has a natural graphical interpretation, which is of independent value. We begin with the following lemma observed before in Figure 7.2.

Lemma 7.7. *Let $\pi = (x, \dots, q, b, y)$ be a discriminating path in a MAG G. Then b and y are connected either via $b \leftrightarrow y$ or $b \rightarrow y$ and in the former case b is a collider on π, in the latter a non-collider.*

Proof. Recall that q is collider and a parent of y and, thus, the edge $q \rightarrow y$ is present and the edge between q and b is either $q \leftarrow b$ or $q \leftrightarrow b$. To prove the claim, we first show that $b \leftarrow y$ cannot occur and distinguish the two ways the edge between q and b is oriented. If $q \leftarrow b$, then we have a directed cycle $q \rightarrow y \rightarrow b \rightarrow q$; if $q \leftrightarrow b$ we would have q as an ancestor of b, which violates the ancestrality property.

For the second part, note that b is always a non-collider if $b \rightarrow y$. In case of $b \leftrightarrow y$, the edge $q \leftarrow b$ cannot occur as b would be an ancestor of y, violating the ancestrality property. Hence, b is a collider in this case. □

Theorem 1.5. *Two MAGs G_1 and G_2 are Markov equivalent if, and only if,*

(I) G_1 and G_2 have the same adjacencies,

(II) G_1 and G_2 have the same unshielded colliders, and

(III) for all edges $b \leftrightarrow y \in G_1$ with $\text{Discr}_{G_1}(b, y) \neq \emptyset$ it holds $b \rightarrow y \notin G_2$ and vice versa.

Proof. We first show that if G_1 and G_2 fulfill the conditions listed above, then they are Markov equivalent by arguing that in this case SRC is satisfied. The first two conditions are identical. Assume the third one holds for the constructive-SRC. Then there is no discriminating path for (x, b, y) with $b \leftrightarrow y$ in G_1 (for the argument we only consider G_1 w.l.o.g.) such that $b \rightarrow y$ in G_2. Hence, it cannot happen that we have a discriminating path π in G_1 and G_2 such that b is a collider in G_1 and a non-collider in G_2 (this is *(iii)* in the SRC). This is because in that case G_2 would have $b \rightarrow y$ by Lemma 7.7.

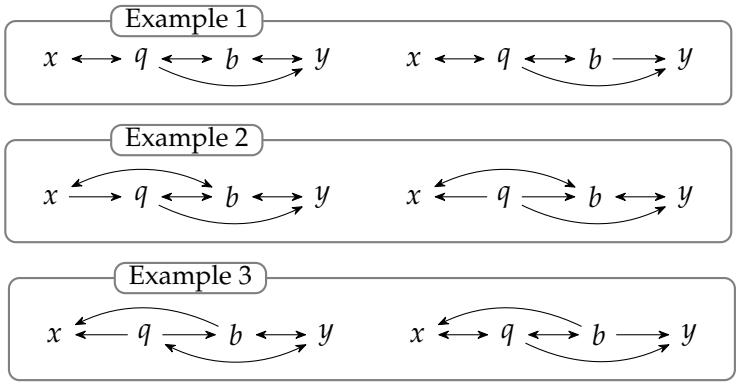


Figure 7.3: Three examples to illustrate the constructive-SRC. Example 2 is from [Ali et al., 2009] and Example 3 is a modification of another example therein.

For the second direction: Assume G_1 and G_2 violate one of the three conditions *(I)*, *(II)* or *(III)*. We show that they are not Markov equivalent. By the SRC, this is obvious for *(I)* and *(II)*. Now consider that *(III)* is violated but *(I)* and *(II)* are true. Then, w.l.o.g., assume that for some $b \leftrightarrow y$ in G_1 there is a discriminating path $\pi = (x, q_1, \ldots, q_p, b, y)$ in G_1 and the edge $b \rightarrow y \in G_2$. It follows by the maximality of G_1 and the fact that x and y are nonadjacent that there is a set Z such that $(x \perp\!\!\!\perp y \mid Z)_{G_1}$. One can easily verify that $q_1, \ldots, q_p \in Z$ and $b \notin Z$. Due to the former observation it holds that $(x \not\perp\!\!\!\perp b \mid Z)_{G_1}$. On the other hand, one can see that $(x \perp\!\!\!\perp y \mid Z)_{G_2}$ and $(x \not\perp\!\!\!\perp b \mid Z)_{G_2}$ cannot both hold in G_2. This is due to the fact that $(x \not\perp\!\!\!\perp b \mid Z)_{G_2}$ immediately implies $(x \not\perp\!\!\!\perp y \mid Z)_{G_2}$ as b is a non-collider not contained in Z. Hence, G_1 and G_2 are not Markov equivalent. □

To illustrate the constructive-SRC, we give three examples (see Figure 7.3) and discuss why or why not Markov equivalence holds (as all considered pairs of graphs have the same adjacencies and unshielded colliders, we focus on whether *(iii)* of the SRC and *(III)* of the constructive-SRC are satisfied).

Example 7.8 (Figure 7.3)**.** *The graphs are not Markov equivalent as the left one contains a discriminating path from x to y with $b \leftrightarrow y$ and the right graph contains the edge $b \rightarrow y$, which violates condition (III) of the constructive-SRC. In the SRC, condition (iii) is not satisfied as the discriminating path $\pi = (x, q, b, y)$ exists in both graphs with b being a collider in the left one and a non-collider in the right one.* ◇

Example 7.9 (Figure 7.3)**.** *The graphs are Markov equivalent. There is a discriminating path (x, q, b, y) in the left graph and it includes the edge $b \leftrightarrow y$, but the right graph also contains the edge $b \leftrightarrow y$ and, hence, (III) does not apply. Accordingly, condition (iii) of the SRC does not apply as (x, q, b, y) is not a discriminating path. An advantage of the constructive-SRC is that one does not have to check for every discriminating path whether it exists in both graphs. It is sufficient to check for the existence of such a path with collider b in one graph, in combination with the edge $b \rightarrow y$ in the other graph.* ◇

Example 7.10 (Figure 7.3)**.** *The graphs are Markov equivalent as well. There is no discriminating path in the left graph, but one in the right graph, namely (x, q, b, y). It contains $b \rightarrow y$ and, hence, (III) does not apply (here, the discriminating path needs to contain $b \leftrightarrow y$ and the* other *graph needs to contain $b \rightarrow y$). Also, (iii) does not apply because, as stated above, there is no discriminating path in the left graph.* ◇

This third example is interesting, because it highlights that *(III)* indeed only refers to discriminating paths with $b \leftrightarrow y$. If then $b \rightarrow y$ in the other graph, one can conclude that Markov equivalence does not hold. If we have a discriminating path with $b \rightarrow y$, even if $b \leftrightarrow y$ in the other graph, we cannot conclude the same. However, as we have seen above, condition *(III)* is not only necessary for Markov equivalence, it is, together with *(I)* and *(II)*, also sufficient. This is because *(iii)* in the SRC could only be violated if we have a discriminating path with a collider in one graph (hence $b \leftrightarrow y$) and a non-collider in the other (hence $b \rightarrow y$) and, consequently, *(III)* would be violated as well. Hence, for the constructive-SRC, it is not necessary to consider discriminating paths with non-colliders b. This entails a simplification, which makes *(III)* easier to check by hand compared to previous formulations (we discuss the algorithmic advantages of the constructive-SRC in the subsequent section) as one only has to look for discriminating path with collider b. Moreover, it also allows to simplify the notion of a discriminating path as *a collider path between non-adjacent x and y for which every vertex but the one before y is a parent of y*.

We note that *(III)* is a generalization of the unshielded collider condition *(ii)*. To see this, we reformulate the criterion for Markov equivalence of DAGs (Theorem 2.4):[5]

Corollary 7.11 (Verma and Pearl [1990], Frydenberg [1990])**.** *Two DAGs G_1 and G_2 are Markov equivalent if, and only if,*

1. *G_1 and G_2 have the same adjacencies and*
2. *if in G_1 there is an unshielded collider $x \rightarrow b \leftarrow y$, then G_2 does not contain $b \rightarrow y$ and vice versa.*

Proof. We argue that *(b)* is true if, and only if, G_1 and G_2 have the same unshielded colliders (implying that Corollary 7.11 is equivalent to Theorem 2.4). The first direction is immediate: if one graph contains the unshielded collider $x \rightarrow b \leftarrow y$ while the other graph orients $b \rightarrow y$, then clearly (x, b, y) is an unshielded collider in only one them.

For the other direction assume w.l.o.g. that G_1 contains an unshielded collider (u, v, w), but G_2 does not. Then G_2 has either $u \leftarrow v$ or $v \rightarrow w$. In both cases *(b)* is violated (set $b = v$ and either $x = w, y = u$ or $x = u, y = w$). □

Corollary 7.12. *Two MAGs G_1 and G_2 are Markov equivalent if, and only if,*

(A) G_1 and G_2 have the same adjacencies,

(B) if there is a collider path $(x, \dots, b, y)$ between non-adjacent x and y with every vertex but x, b and y being a parent of y in G_1, then G_2 does not contain the edge $b \rightarrow y$ and vice versa.

Proof. The collider path $(x, \dots, b, y)$ may only consist of three vertices, i.e., it could be an unshielded collider. If the other graph were to contain the edge $b \rightarrow y$, then it would not have that same collider, meaning the graphs are not Markov equivalent by *(II)*. If the collider path consists of more than three vertices, the formulation equals *(III)*. □

We remark that this corollary applies only to MAGs without undirected edges (in contrast to the constructive-SRC). However, only minor modifications are necessary to handle undirected edges as well. We discuss these in Section 7.6.

[5] There are even further formulations of *(III)*, e.g., in terms of parametrizing sets, as pointed out by an anonymous reviewer: If there is a discriminating path for $\{x, b, y\}$ with non-collider b, then the set is parametrizing in both graphs.

7.4 Algorithm for Testing Markov Equivalence of MAGs

In the previous section, we derived a simple characterization of Markov equivalence for MAGs. In this section, we deal with the computational side of the problem and discuss how this new characterization can be tested. The algorithm we propose has a worst-case run-time of $O(n^3)$, thus being significantly faster than previous approaches. Moreover, for sparse graphs, which are very common in causal modeling, we even report linear time in the number of vertices.

We check the conditions *(I)* and *(II)* naively. For checking the third condition *(III)*, we need to test for each $b \leftrightarrow y$ in G_k with $k \in \{1,2\}$, for which $b \to y$ is an edge in the other graph $G_{k'}$ (with $k' = 3 - k$), whether there is a discriminating path for b and y. We do this by considering every choice of y consecutively, computing for each the bidirected connected components of its parents (we call these the *parent districts*) that support our computations.

Definition 7.13 (Parent districts)**.** *Given a MAG G and a vertex y, the bidirected connected components of $G[Pa(y)]$ are termed the* parent districts *of y and denoted as $\mathcal{D}(y)$.*

This notion is useful as the middle part of a discriminating path consists solely of such vertices $q_1, \ldots, q_p$ in a single parent district of y. Once the parent districts have been computed, one can check if, for a certain district $D \in \mathcal{D}(y)$, there is a vertex x non-adjacent to y and a parent or sibling of D, which can function as the start of the discriminating path. If this is the case, it remains to consider all vertices b which are siblings of D and y. For these, we can conclude that they are part of a discriminating path $x \mathrel{*\!\!\to} q_1 \leftrightarrow \ldots \leftrightarrow q_p \leftrightarrow b \leftrightarrow y$. If $b \to y$ in the other graph, the graphs are not Markov equivalent. Figure 7.4 illustrates this approach and Algorithm 7.1 gives an implementation.

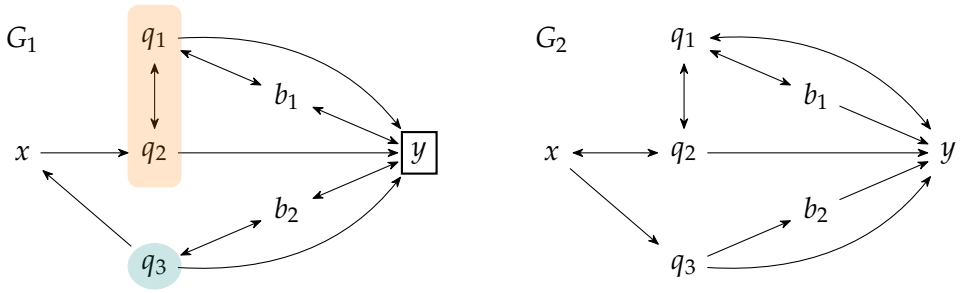


Figure 7.4: Illustration of Algorithm 7.1 checking vertex $\boxed{y}$ in G_1 (line 5) with $\mathcal{D}(y) = \{$ $\{q_3\}$, $\{q_1, q_2\}$ $\}$. For $D = \{q_3\}$, the set $Pa_{G_1}(D) \cup Si_{G_1}(D) \setminus Ne_{G_1}(y)$ is empty as x is a child of D. Hence, Algorithm 7.1 does not consider D further (line 8). For $D = \{q_1, q_2\}$, the set $Pa_{G_1}(D) \cup Si_{G_1}(D) \setminus Ne_{G_1}(y)$ contains x, which is a parent of D but not a neighbor of y. Moreover, b_1 is a sibling of both D and y. Hence, we obtain the discriminating path $x \to q_2 \leftrightarrow q_1 \leftrightarrow b_1 \leftrightarrow y$. As $b_1 \to y$ in G_2, the algorithm reports that the graphs are not Markov equivalent. Note that for SRC *(iii)* is violated due to the discriminating path x, q_2, q_1, y in both graphs with G_1 containing non-collider $q_2 \leftrightarrow q_1 \to y$ and G_2 containing collider $q_2 \leftrightarrow q_1 \leftrightarrow y$. The discriminating path for $q_1 \leftrightarrow y$ in G_2 and the corresponding edge $q_1 \to y$ would also be detected by Algorithm 7.1. Note that here $\{q_2\}$ is a parent district of y (q_1 is not part of this district as it is not a parent of y in G_2).

Algorithm 7.1: Checking the constructive-SRC.

input : Two MAGs $G_1 = (V_1, A_1, B_1)$, $G_2 = (V_2, A_2, B_2)$.
output: Whether G_1 and G_2 are Markov equivalent.

```
1  if (I) or (II) of the constructive-SRC is violated then
2  |  return Not Markov equivalent.
3  end
4  foreach G_k with k ∈ {1,2} do
5  |  foreach y ∈ V_k do
6  |  |  foreach D ∈ 𝒟(y) do
7  |  |  |  Compute Pa_{G_k}(D) and Si_{G_k}(D).
8  |  |  |  if Pa_{G_k}(D) ∪ Si_{G_k}(D) \ Ne_{G_k}(y) ≠ ∅ then
9  |  |  |  |  foreach b ∈ Si_{G_k}(D) ∩ Si_{G_k}(y) do
10 |  |  |  |  |  Let G_{k'} be the other graph, i.e., k' = 3 − k.
11 |  |  |  |  |  if b → y in G_{k'} then
12 |  |  |  |  |  |  return Not Markov equivalent.
13 |  |  |  |  |  end
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 return Markov equivalent.
```

Theorem 7.14. *Algorithm 7.1 checks whether two MAGs are Markov equivalent in time $O(n^3)$ for general graphs and expected time $O(n \cdot \Delta^2)$ for graphs with maximal degree Δ.*

Proof. For the correctness of Algorithm 7.1, we need to show that *(III)* of the constructive-SRC is correctly checked. If the algorithm returns "*not Markov equivalent*" in line 13, then there exists a b and y such that $b \leftrightarrow y$ in one graph and $b \rightarrow y$ in the other. Moreover, in the former graph there exists a parent district $D \in \mathcal{D}(y)$ such that there is a $x \in Pa_{G_k}(D) \cup Si_{G_k}(D) \setminus Ne_{G_k}(y)$ (this set is non-empty in line 8) and it is guaranteed that b is not only a sibling of y, but also of D. Hence, there is a discriminating path $x \ast\!\!\rightarrow q_1 \leftrightarrow \ldots \leftrightarrow q_p \leftrightarrow b \leftrightarrow y$, with $q_1, \ldots, q_p \in D$ and q_1 being the sibling/child of x and q_p being the sibling of b. The collider path from q_1 to q_p exists by the definition of D. For the same reason, we have that $q_1, \ldots, q_p$ are parents of y. Note, in particular, that $b \neq x$ and both are not in D. Thus, *(III)* is violated and the output is correct.

For the other direction, if the graph contains a violation of *(III)*, then there is a discriminating path for $b \leftrightarrow y$ (while the other graph contains $b \rightarrow y$). The existence of such a path is detected as all discriminating paths for $b \leftrightarrow y$ have the form $x \ast\!\!\rightarrow q_1 \leftrightarrow \ldots \leftrightarrow q_p \leftrightarrow b \leftrightarrow y$ with $q_1, \ldots, q_p$ being parents of y. Thus, there is a parent district of y, which has x as parent/sibling and b as a sibling. Hence, the algorithm outputs "*not Markov equivalent*" in line 13.

Regarding the run-time, note that checking *(I)* and *(II)* in line 1 is possible in time $O(n^2)$, respectively $O(n^3)$. If the graph is sparse the run-times $O(n\Delta)$, resp. $O(n\Delta^2)$, follow. For the latter case consider for each vertex all pairs of its parents and test whether

they are adjacent.[6]) For checking *(III)*, there are n vertices y considered per graph at line 5. Computing the parent districts for one y can be done in time $O(n^2)$ or $O(\Delta^2)$ if Δ is the maximal degree of the graph, as finding the connected components of a (sub)graph with s vertices takes $O(s^2)$ time in the worst-case. Hence, this step can be performed in time $O(n^3)$ or $O(n \cdot \Delta^2)$. Also the neighbors/parents/siblings of y and all its parent districts may be precomputed in this phase as well.

Further, there are two nested for loops, one over the parent districts (there are at most $O(n)$ or $O(\Delta)$ many) and one over b (again there are $O(n)$ or $O(\Delta)$ choices for b)[7]. Finally line 11 can be performed in (expected) time $O(1)$ (see footnote 6), yielding again $O(n \cdot \Delta^2)$ proving the claim. □

We conclude that for graphs with maximal degree Δ the expected run-time can be written as $O(n \cdot \Delta^2)$, which is linear in the number of vertices for a constant Δ. We note that this is a significant improvement over Hu and Evans [2020], who reported time $O(m^2) = O(n^2)$ for sparse random graphs.

7.5 A Different Approach to Markov Equivalence Testing

The continual improvement of algorithms for testing the Markov equivalence of MAGs from exponential time (SRC) over $O(n^9)$ (Ali et al. [2009]), $O(n^5)$ (Hu and Evans [2020]) and $O(n^4)$ (Claassen and Bucur [2022]) to $O(n^3)$ begs the question of what the best achievable run-time is. Is it possible to test Markov equivalence of MAGs in $O(n^2)$? In particular, for DAGs this run-time *can* be achieved by building on the fact that their corresponding CPDAG can be constructed efficiently.

More precisely, to test whether two DAGs are Markov equivalent, one may compute the corresponding CPDAGs C_1 and C_2 and check whether $C_1 = C_2$. The complexity of this approach hinges on the complexity of converting DAGs to CPDAGs, which can be performed in linear-time through the algorithm given by Chickering [1995]. There is also a less-clever strategy to do this by "imitating" the PC algorithm. That is initializing C as the skeleton of D, next setting all v-structures of D in C and finally applying the Meek rules. However, this approach yields a worse run-time.

Coming back to MAGs, we note that this second and slower approach can be used in this setting as well. For a MAG G, one can imitate the FCI algorithm [Spirtes et al., 2000], which is the counterpart of the PC algorithm under latent confounding/selection bias, to obtain its corresponding *partial ancestral graph* (PAG) [Zhang, 2008a], which is, analogously to the CPDAG for DAGs, a compact and unique representation of an equivalence class. This is done by first initializing P as the skeleton of G, setting the unshielded colliders according to G and, finally, applying the ten completion rules given by Zhang [2008b] (see also Ali et al. [2005]). This approach yields a polynomial-time algorithm for testing Markov equivalence of MAGs, but with a rather large polynomial:[8] One can compute the PAGs P_1, P_2 for the given MAGs G_1, G_2 and check whether they are identical[9].

The other strategy currently cannot be translated to MAGs as there is no counterpart for the DAG-to-CPDAG algorithm to directly transform a MAG into a PAG. Hence, a

[6] We can perform adjacency tests in $O(1)$ by storing the graph as adjacency matrix. For sparse graphs we may avoid $O(n^2)$ space by using hash tables, which yields expected time $O(1)$.

[7] Forming $Si_{G_k}(D) \cap Si_{G_k}(y)$ can be done in $O(n)$. Expected time $O(\Delta)$ can be obtained using hash tables.

[8] The time is a polynomial of order roughly $O(m^3 \cdot n)$ as for every undirected edge we have to check whether global conditions hold (Zhang [2008b] briefly discuss the run-time, mentioning $O(n \cdot m)$ for checking the fourth rule for a single edge.)

[9] This strategy has some parallels to Ali et al. [2009], due to the fact that colliders with order also play a key role in the completeness of the FCI rules [Ali et al., 2005].

better understanding of PAGs might be needed for further progress and we deem this as an important topic for future research.

7.6 Related Problems

So far, our focus lied on the problem of testing Markov equivalence of MAGs without undirected edges. In this section we discuss the connection to more general formulations of the problem. First, we note that the constructive-SRC and Algorithm 7.1 also work for MAGs with undirected edges. This is because the SRC also holds in this setting and that there cannot be an undirected edge in a discriminating path (in particular, the edge between b and y cannot be undirected). For Corollary 7.12 a modification is necessary: condition *(II)* has to be changed to "If there is a collider path $x, \dots, b, y$ between non-adjacent x and y with every vertex but x, b and y being a parent of y in one graph, then the other graph does neither contain the edge $b \to y$ *nor* the edge $b - y$." This is necessary as a collider $u \ast\!\!\to v \leftarrow\!\!\ast\, w$ in one graph might correspond to a non-collider $u - v - w$ in the other graph – and these graphs are, of course, not Markov equivalent.

Further related problems are obtained by removing the maximality or the ancestrality requirement (or both). In that case, we deal with general *acyclic directed mixed graphs* (ADMGs). These are graphs that may contain directed and bidirected edges with the only requirement that there is no directed cycle. The SRC and constructive-SRC do not apply for ADMGs as they explicitly use the maximality and ancestrality properties. However, one can transform ADMGs into equivalent MAGs and, thus, test the Markov equivalence of ADMGs using the algorithms for MAGs. As it turns out, the currently fasted algorithm for the ADMG-to-MAG transformation (Algorithm 2 in Hu and Evans [2020]) requires time $O(n^4)$ and is, thus, the bottleneck in this approach (testing the equivalence of MAGs is in $O(n^3)$ by Theorem 7.14).

It is unclear to us whether this transformation can be performed in $O(n^3)$. A central part of it involves the computation of so-called *inducing paths*, where it has to be checked for every pair of vertices (x, y) whether there is a collider path between x and y via vertices in $An(x, y)$. Since we have $O(n^2)$ such pairs and since, further, graph traversal is in $\Omega(n + m)$, this direct approach necessarily produces a workload of $O(m \cdot n^2)$. We believe that it will be central for the development of faster ADMGs equivalence tests to better understand the complexity of ADMG-to-MAG and consider this as an interesting question for further work.

7.7 Experimental Evaluation

To emphasize the practical effectiveness of the constructive-SRC and, in particular, Algorithm 7.1, we compare it experimentally with the algorithm proposed by Hu and Evans [2020] on synthetic data. Both algorithms were implemented in the Julia programming language [Bezanson et al., 2017] and we ran the experiments on a desktop computer with an Intel(R) Core(TM) i7-8565U CPU and 16GBs of RAM.[10] Synthetic MAGs were generated with the process described in [Hu and Evans, 2020]: Fix a topological ordering τ of the vertices, then add k edges uniformly at random, and finally direct each edge with probability $1/2$ according to τ, else leave it undirected. Finally, replacing remaining undirected edges with bidirected edges yields an ADMG, which can in turn be transformed into a MAG, as discussed in Section 7.6.

[10] The code is available under `https://github.com/mwien/magequivalence`.

Table 7.1: Distribution of directed edges ($\rightarrow$) and bidirected edges ($\leftrightarrow$) in the randomly generated ADMGs and in the corresponding MAGs. For every row we generated 250 random ADMGs with n vertices and $3n$ edges, and show the average of directed or bidirected edges they contain.

	ADMG			MAG		
n	$\rightarrow$	$\leftrightarrow$	$\rightarrow$ / $\leftrightarrow$	$\rightarrow$	$\leftrightarrow$	$\rightarrow$ / $\leftrightarrow$
250	373.844	376.156	0.9962	394.38	401.88	0.9840
500	752.536	747.464	1.0081	772.148	771.456	1.0022
750	1125.812	1124.188	1.0023	1146.012	1147.944	0.9991
1000	1500.308	1499.692	1.0010	1519.616	1523.628	0.9980
1250	1873.564	1876.436	0.9990	1892.58	1899.968	0.9966
1500	2251.344	2248.656	1.0016	2270.228	2272.872	0.9992
1750	2627.564	2622.436	1.0023	2647.156	2646.568	1.0005
2000	3002.328	2997.672	1.0018	3021.66	3021.5	1.0003

For a fair comparison with the experiments in [Hu and Evans, 2020], we run a modified version of Algorithm 7.1. It generates, for *a single MAG*, a set of all adjacencies A (for checking *I*), a set of all v-structures V (for checking *II*), and the set C of all $b \leftrightarrow y$ that are part of a discriminating path, as well as the set N of all $b \rightarrow y$ (for checking *III*). Clearly, if one were to generate these sets for two MAGs G_1 and G_2, testing Markov equivalence would reduce to checking whether $A_1 = A_2$, $V_1 = V_2$, $C_1 \cap N_2 = \emptyset$, and $C_2 \cap N_1 = \emptyset$.

This approach provides a finer control over the experiments as it avoids the possibility of an "early stopping" at line 1 or line 13 of Algorithm 7.1 (which can happen if the given MAGs are not equivalent and would give an unfair advantage to our algorithm). It also enables us to consider single MAGs, which are simpler to generate randomly than, e. g, two random Markov equivalent MAGs. The reported run-times can be viewed, for our algorithm as well as for [Hu and Evans, 2020], as essentially half the time occurring when two Markov equivalent DAGs are compared (because these steps have to be performed for both graphs).

As in Hu and Evans [2020], we generate graphs with $k = 3n$ edges. Note that this is the number of edges in the generated ADMG and not in the MAGs on which the algorithms run. The transformation of an ADMG into a MAG might generate new edges, this is documented in Table 7.1. For our experiments, we ran both algorithms on the same 250 randomly generated graphs for each choice of n and report the average time they used in Figure 7.5. It can be seen that Algorithm 7.1 is faster in all settings. We can also observe that for larger graphs, the advantage increases – which implies that the algorithm in fact has a better asymptotic behavior. Finally, the absolute run-time of Algorithm 7.1 is generally extremely low (for the considered inputs only fractions of a second[11]).

7.8 Conclusions

The problem of characterizing and checking Markov equivalence of MAGs is one of the most basic problems for this model class. Through the above results, we have at our dis-

[11]Generating significantly harder instances is not a trivial task as the random generation process relies on the ADMG-to-MAG task, which currently cannot be performed faster than in $O(n^4)$.

	Algorithm HE		Algorithm C-SRC	
n	Avg. Time	Std. Dev.	Avg. Time	Std. Dev.
250	0.0487s	0.0101	0.0015s	0.0009
500	0.1058s	0.0388	0.0032s	0.0051
750	0.1605s	0.0279	0.0049s	0.0065
1000	0.2587s	0.0594	0.0062s	0.0058
1250	0.3579s	0.0684	0.0085s	0.0081
1500	0.4629s	0.0789	0.0091s	0.0058
1750	0.5373s	0.0626	0.0106s	0.0021
2000	0.6794s	0.0778	0.0119s	0.0024

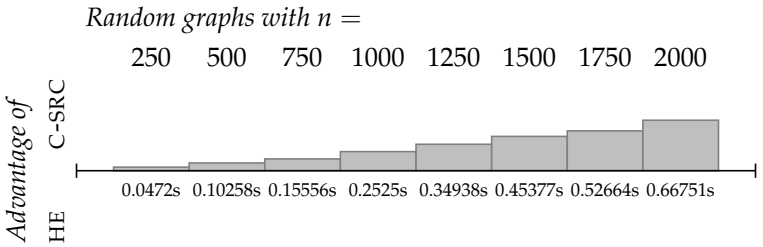


Figure 7.5: *Advantage plot* that compares our implementation (C-SRC) with the algorithm by Hu and Evans (HE). Each bar corresponds to an experiment on random graphs with n vertices (denoted above the bars) and $3n$ edges. For each experiment we measured the average time needed by both algorithms over 250 instances. If C-SRC uses t_1 seconds and HE took t_2 seconds, then the *advantage* of C-SRC over HE is defined by $t_2 - t_1$ (i.e., the advantage is positive iff C-SRC is faster). The advantage (in seconds) is shown below the bars.

posal a simple criterion for Markov equivalence as well as an efficient algorithm. However, a large number of problems are still partially or fully unsolved.

This includes the problems of enumerating all MAGs in a Markov equivalence class as well as counting them or sampling from them – problems we tackled for DAGs in this thesis. Indeed, the only such problem which *is* solved is the extension problem for PAGs, which can be solved in linear-time $O(n + m)$, as shown by Zhang [2008b], in analogy to the CPDAG extension task discussed in Section 2.3. However, already incorporating background knowledge into the PAG model is a fundamental challenge. Hence, this model class is an interesting avenue for further algorithmic research in the context of Markov equivalence.

8

Conclusions

In this thesis, we have tackled fundamental algorithmic problems in the context of Markov equivalence. The results effectively conclude the complexity-theoretical study of the various aspects (FINDING, LISTING and COUNTING) of computational problems in the context of consistent extensions introduced in Chapter 1: In case of FINDING, we precisely capture its hardness by giving matching lower and upper bounds conditional on the Strong-Triangle-Conjecture. For LISTING, we give an asymptotically optimal[1] $O(n+m)$ delay algorithm. Finally, the main achievement of this thesis – the Clique-Picking algorithm – certifies that COUNTING is possible in polynomial-time for CPDAGs, while we show #P-hardness in the case of general PDAGs.

Our algorithms are not only of theoretical nature, but implementable and practical. The implementations of the algorithms presented in this work are publicly and freely available (links to GitHub repositories are given in the individual chapters) and also being merged into the Julia package `CausalInference.jl` [Schauer et al., 2023]. In Chapter 6, we discussed applications of the Clique-Picking approach, in particular to sample uniformly from an MEC and to handle interventional CPDAGs. The results are relevant in the field of active learning, that is efficiently designing experiments to recover the true underlying DAG, as well as causal effect identification over MECs.

While the algorithmic and graph-theoretical properties of Markov equivalence are well-studied in the case of DAGs, many fundamental tasks remain open in the case of more expressive causal models such as MAGs. In this thesis, we tackled the problem of testing Markov equivalence of MAGs, giving a new criterion and a more efficient algorithm improving the state-of-the-art to $O(n^3)$. However, many algorithmic tasks going beyond this remain unexplored to a large degree. This concerns for example the tasks of counting the number of Markov equivalent MAGs and listing them [Malinsky and Spirtes, 2016].

One of the reasons that MAGs are less-well understood compared to DAGs is that the theory behind them is significantly more intricate. Generally, the goal should not only be to decrease the complexity gap between MAGs and DAGs, as this might not always be possible, but to also characterize it, for example through conditional lower bounds such as the ones shown in Chapter 3. The task of testing Markov equivalence could be a fitting example for this kind of study. In case of two DAGs, it can be solved in linear-time $O(n+m)$, simply by transforming each into the corresponding CPDAG using the algorithm by Chickering [1995] and afterwards testing equality of the resulting graphs,

[1] The algorithm is optimal if each DAG is output separately. If one were to allow in-place modification, better run-times might be possible.

while the best upper bound is $O(n^3)$ for MAGs by the results in Chapter 7 as mentioned above. Recently, the same problem has been studied for *cyclic* graphs (without allowing for unobserved confounders) and an $O(n^5)$ algorithm has been proposed [Claassen and Mooij, 2023]. It would be interesting to derive lower bounds that formalize this hierarchy from a computational perspective.

Bibliography

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

R. Ayesha Ali, Thomas S. Richardson, Peter Spirtes, and Jiji Zhang. Towards characterizing Markov equivalence classes for directed acyclic graphs with latent variables. In *Conference on Uncertainty in Artificial Intelligence, UAI '05*, pages 10–17, 2005.

R. Ayesha Ali, Thomas S. Richardson, and Peter Spirtes. Markov equivalence for ancestral graphs. *The Annals of Statistics*, 37(5B):2808–2837, 2009.

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.

Ilkka Autio, Olaf Laczak, and Kari Vasko. On the effective computation of the size of the Markov equivalence class of a Bayesian belief network structure. *Technical Report*, 1999. [Accessed online 5-11-2022, `citeseerx.ist.psu.edu/pdf/4ecbd6e0cc6c6266dc4c5e25a01edc17357294ca`].

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.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. URL `https://doi.org/10.1137/141000671`.

Jean R. S. 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 *Annual ACM Symposium on Theory of Computing, STOC '91*, pages 175–181, 1991.

David M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Conference on Uncertainty in Artificial Intelligence, UAI '95*, pages 87–98, 1995.

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

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

Tom Claassen and Ioan G. Bucur. Greedy equivalence search in the presence of latent confounders. In *Conference on Uncertainty in Artificial Intelligence, UAI '22*, pages 443–452. PMLR, 2022.

Tom Claassen and Joris M. Mooij. Establishing Markov equivalence in cyclic directed graphs. In *Conference on Uncertainty in Artificial Intelligence, UAI '23*, pages 433–442. PMLR, 2023.

Dorit Dor and Michael Tarsi. A simple algorithm to construct a consistent extension of a partially oriented graph. *Technicial Report R-185, Cognitive Systems Laboratory, UCLA*, 1992.

Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

Frederick Eberhardt. Almost optimal intervention sets for causal discovery. In *Conference in 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 *Conference in Uncertainty in Artificial Intelligence, UAI '05*, pages 178–184. AUAI Press, 2005.

Morten Frydenberg. The chain graph Markov property. *Scandinavian Journal of Statistics*, pages 333–353, 1990.

D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

Robert Ganian, Thekla Hamm, and Topi Talvitie. An efficient algorithm for counting Markov equivalent DAGs. In *AAAI Conference on Artificial Intelligence, AAAI '20*, pages 10136–10143, 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 *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 *AAAI Conference on Artificial Intelligence, AAAI '19*, pages 3664–3671, 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.

Chris Godsil and Gordon F. Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2001.

Kristjan H. Greenewald, Dmitriy Katz, Karthikeyan Shanmugam, Sara Magliacane, Murat Kocaoglu, Enric B. Adserà, and Guy Bresler. Sample efficient active learning of causal trees. In *Advances in 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, 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.

Zhongyi Hu and Robin Evans. Faster algorithms for Markov equivalence. In *Conference on Uncertainty in Artificial Intelligence, UAI '20*, pages 739–748. PMLR, 2020.

David S. Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

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

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of statistical software*, 47:1–26, 2012.

Donald E. Knuth. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2014.

Steffen L. Lauritzen. *Graphical models*. Clarendon Press, 1996.

Steffen L. Lauritzen and Nanny Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17:31–57, 1989.

Nathan Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.

Malte Luttermann, Marcel Wienöbst, and Maciej Liśkiewicz. Practical algorithms for orientations of partially directed graphical models. In *Conference on Causal Learning and Reasoning, CLeaR '23*, volume 213 of *Proceedings of Machine Learning Research*, pages 259–280. PMLR, 11–14 Apr 2023.

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.

Daniel Malinsky and Peter Spirtes. Estimating causal effects with ancestral graph Markov models. In *Conference on Probabilistic Graphical Models, PGM '16*, pages 299–309. PMLR, 2016.

David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

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

Joris M. Mooij and Tom Heskes. Cyclic causal discovery from continuous equilibrium data. In *Conference on Uncertainty in Artificial Intelligence, UAI '13*, pages 431–439, 2013.

Torsten Mütze. Combinatorial Gray codes-an updated survey. *arXiv preprint arXiv:2202.01280*, 2022.

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

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

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

Judea Pearl, Madelyn Glymour, and Nicholas P. Jewell. *Causal inference in statistics: A primer*. John Wiley & Sons, 2016.

Emilija Perković. Identifying causal effects in maximally oriented partially directed acyclic graphs. In *Conference on Uncertainty in Artificial Intelligence, UAI '20*, volume 124 of *Proceedings of Machine Learning Research*, pages 530–539. AUAI Press, 2020.

Emilija Perković, Markus Kalisch, and Marloes H. Maathuis. Interpreting and using cpdags with background knowledge. In *Conference on Uncertainty in Artificial Intelligence, UAI '17*. AUAI Press, 2017.

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.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

Joseph D. Ramsey, Kun Zhang, Madelyn Glymour, Ruben S. Romero, Biwei Huang, Imme Ebert-Uphoff, Savini Samarasinghe, Elizabeth A. Barnes, and Clark Glymour. Tetrad—a toolbox for causal discovery. In *8th International Workshop on Climate Informatics*, page 29, 2018.

Thomas Richardson and Peter Spirtes. Ancestral graph Markov models. *The Annals of Statistics*, 30(4):962–1030, 2002.

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

Carla Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39(4):605–629, 1997.

Moritz Schauer and Marcel Wienöbst. Causal structure learning with momentum: Sampling distributions over Markov equivalence classes of dags. *arXiv preprint arXiv:2310.05655*, 2023.

Moritz Schauer, Martin Keller, and Marcel Wienöbst. Causalinference.jl (v0.10). `https://github.com/mschauer/CausalInference.jl`, 2023.

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

Marco Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3), 2010.

Milan Sekanina. On an ordering of the set of vertices of a connected graph. *Publ. Fac. Sci. Univ. Brno*, 412:137–142, 1960.

Oylum Seker, Pinar Heggernes, Tínaz Ekim, and Z. Caner Taskin. Linear-time generation of random chordal graphs. In *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 *Advances in Neural Information Processing Systems, NIPS '15*, pages 3195–3203, 2015.

Vidya Sagar Sharma. Counting background knowledge consistent Markov equivalent directed acyclic graphs. In *Conference on Uncertainty in Artificial Intelligence, UAI '23*, volume 216 of *Proceedings of Machine Learning Research*, pages 1911–1920. PMLR, 2023.

Ilya Shpitser and Judea Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *National Conference on Artificial Intelligence, AAAI '06*, volume 21(2), pages 1219–1226. AAAI Press, 2006.

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

Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, 1989.

Peter Spirtes and Thomas Richardson. A polynomial time algorithm for determining DAG equivalence in the presence of latent variables and selection bias. In *International Workshop on Artificial Intelligence and Statistics, AISTATS '97*, pages 489–500, 1997.

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

Chandler Squires. *causaldag: creation, manipulation, and learning of causal models*, 2018. URL `https://github.com/uhlerlab/causaldag`.

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

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

Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

Topi Talvitie and Mikko Koivisto. Counting and sampling Markov equivalent directed acyclic graphs. In *AAAI Conference on Artificial Intelligence, AAAI 19*, pages 7984–7991, 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.

Johannes Textor, Benito Van der Zander, Mark S. Gilthorpe, Maciej Liśkiewicz, and George T. H. Ellison. Robust causal inference using directed acyclic graphs: the R package 'dagitty'. *International Journal of Epidemiology*, 45(6):1887–1894, 2016.

Benito van der Zander and Maciej Liśkiewicz. Separators and adjustment sets in Markov equivalent dags. In *AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 3315–3321, 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.

Lieven Vandenberghe and Martin S. Andersen. Chordal graphs and semidefinite optimization. *Foundations and Trends® in Optimization*, 1(4):241–433, 2015.

Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *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 *Conference on Uncertainty in Artificial Intelligence, UAI '92*, pages 323–330. Elsevier, 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.

Dominic Welsh. The Tutte polynomial. *Random Structures & Algorithms*, 15(3-4):210–228, 1999.

Douglas B. West. *Introduction to graph theory*. Prentice Hall, 2001.

Marcel Wienöbst. On the computational complexity of graph moralization. *Blog post at mwien.github.io*, 2023.

Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Extendability of causal graphical models: Algorithms and computational complexity. In *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 *AAAI Conference on Artificial Intelligence, AAAI '21*, volume 35, pages 12198–12206. AAAI Press, 2021b.

Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. A new constructive criterion for Markov equivalence of MAGs. In *Conference on Uncertainty in Artificial Intelligence, UAI '22*, volume 180 of *Proceedings of Machine Learning Research*, pages 2107–2116. PMLR, 2022.

Marcel Wienöbst, Malte Luttermann, Max Bannach, and Maciej Liśkiewicz. Efficient enumeration of Markov equivalent dags. In *AAAI Conference on Artificial Intelligence, AAAI '23*, volume 37, pages 12313–12320, 2023.

Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Polynomial-time algorithms for counting and sampling Markov equivalent dags with applications. *Journal of Machine Learning Research*, 24(213):1–45, 2023.

Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.

Jiji Zhang. Causal reasoning with ancestral graphs. *Journal of Machine Learning Research*, 9:1437–1474, 2008a.

Jiji Zhang. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence*, 172(16-17):1873–1896, 2008b.

Jiji Zhang and Peter Spirtes. A transformational characterization of Markov equivalence for directed acyclic graphs with latent variables. In *Conference on Uncertainty in Artificial Intelligence, UAI '05*, pages 667–674, 2005.

Hui Zhao, Zhongguo Zheng, and Baijun Liu. On the Markov equivalence of maximal ancestral graphs. *Science in China Series A: Mathematics*, 48(4):548–562, 2005.

List of Symbols and Notation

Notation	Reference or Description
n	number of vertices
m	number of edges
V	set of vertices
E	set of undirected edges
A	set of directed edges (arcs)
B	set of bidirected edges
$Ne(v)$	set of neighbors of vertex v
$Un(v)$	set of *undirected* neighbors of vertex v
$Pa(v)$	set of parents of vertex v
$Ch(v)$	set of children of vertex v
$Si(v)$	set of siblings of vertex v
$An(v)$	set of ancestors of vertex v
$De(v)$	set of descendants of vertex v
$x \sim y$	denotes that vertices x and y are adjacent
$\delta^-(v)$	number of parents of vertex v
$\delta^+(v)$	number of children of vertex v
$\delta(v)$	number of undirected neighbors of vertex v
$\Delta(v)$	number of neighbors of v
$\Delta(G)$	maximum number of neighbors of a vertex in graph G
$U(G)$	undirected subgraph of graph G
$\mathcal{C}(G)$	connected components of (undirected subgraph of) G
$\tau + \pi$	concatenation of sequences τ and π
$x \preceq y$	x precedes y in a linear or partial ordering
$G[S]$	induced subgraph of G over vertex set S
$G[\tau]$	orientation of G according to linear ordering τ
$\Sigma(G)$	set of minimal separators of graph G
$\Pi(G)$	set of maximal cliques of graph G
$\Pi^v(G)$	set of maximal cliques of graph G, which contain vertex v
$\Pi^S(G)$	set of maximal cliques of graph G, which contain vertex set S
$X_a \perp\!\!\!\perp_{\mathbb{P}} X_b$	conditional independence of random variables X_a and X_b in $\mathbb{P}$
$a \perp\!\!\!\perp_G b$	d-separation of vertices a and b in graph G
$[D]$	for a DAG D, it denotes the set of Markov equivalent DAGs to D
$[G]$	for a CPDAG G, it denotes the MEC of DAGs represented by G

$\text{pdag}(\mathcal{C})$	Definition 2.6
$\text{EXT}(G)$	set of consistent extensions of graph G
$\text{AMO}(G)$	set of acyclic moral extensions of undirected graph G
$P_i(v)$	the number of neighbors of v which are visited at step i of an MCS
M_i	the set of vertices with largest $P_i(v)$ at step i of an MCS
M_i^A	the set of vertices with largest $P_i(v)$ with no incoming arcs in A from an unvisited vertex at step k of an MCS
$\alpha(s)$	$\lvert\{ x - y \mid x - v \in E \wedge v - y \in E \wedge x \sim_G y \}\rvert$
$\beta(s)$	$\lvert\{ x - y \mid x - v \in E \wedge y \to v \in A \wedge x \sim_G y \}\rvert$
$B(G)$	bucket graph of G (Definition 3.22)
G^s	Definition 5.5
G^σ and G^S	Definition 5.10
$\text{poly}(n)$	polynomial in n
T_k	set of vertices, which were output before or at step k of Algorithm 5.1
$t(v)$	step of Algorithm 5.1 at which v is output
$\phi(S)$	Definition 5.19
$\phi_{T^R}(S)$	Definition 5.30
$\rho(X)$	Definition 5.35
$[D]_\mathcal{I}$	$\mathcal{I}$-MEC of DAG D
$\mathcal{E}_{\mathcal{I}(D)}$	interventional CPDAG of $[D]_\mathcal{I}$, that is $\text{pdag}([D]_\mathcal{I})$
θ_{ji}	causal effect of X_i on X_j
$\phi'_{T^R}(S)$	Definition 6.10
$Dis(v)$	set of vertices in the same district as vertex v
$Discr_G(b, y)$	set of all discriminating paths $\pi = (x, q_1, \ldots, q_p, b, y)$ for b
$\mathcal{D}(y)$	Definition 7.13

Curriculum Vitae

Persönliche Daten

Marcel Wienöbst
geboren 1995 in Hameln

Akademischer Werdegang

2019-heute	**Promotion**, *Institut für Theoretische Informatik, Universität zu Lübeck* Thema: Algorithms for Markov Equivalence. Betreuer: Maciej Liśkiewicz
2016-2019	**Master Informatik**, *Universität zu Lübeck*, Abschlussnote 1,0 Masterarbeit: Constraint-based causal structure learning exploiting low-order conditional independencies. Betreuer: Maciej Liśkiewicz.
2013-2016	**Bachelor Informatik**, *Universität zu Lübeck*, Abschlussnote 1,4 Bachelorarbeit: Experimentelle Analyse von Algorithmen zur Lösung des Bisektionsproblems in Graphen. Betreuer: Martin Schuster, Maciej Liśkiewicz.
2013	**Abitur**, *KGS Salzhemmendorf*, Note 1,0

Wissenschaftliche Tätigkeiten

2019-heute	**Wissenschaftlicher Mitarbeiter**, *Institut für Theoretische Informatik, Universität zu Lübeck*
zw. 2015 u. 2019	**Wissenschaftliche Hilfskraft**, *Universität zu Lübeck* Darunter im DFG-Projekt *Causality: an algorithmic framework and a computational complexity perspective* von Maciej Liśkiewicz.

Weiteres Engagement

2019-heute	**Coach/Organisator bei den ICPC-Wettbewerben**, *Universität zu Lübeck* Jurymitglied beim Wintercontest 2020, 2022, 2023 und GCPC 2022, 2023.
2015-2019	**Teilnehmer an den ICPC-Wettbewerben**, *Universität zu Lübeck* Platz 9 (Bronzemedaille) beim GCPC 2018 und Platz 28 beim NWERC 2017.

Auszeichnungen

Best Student Paper Award (UAI 2022) für das Paper
A New Constructive Criterion for Markov Equivalence of MAGs.

Best Student Paper Award (UAI 2021) für das Paper
Extendability of Causal Graphical Models: Algorithms and Computational Complexity.

Distinguished Paper Award (AAAI 2021) für das Paper
Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs.

Top Reviewer Award (UAI 2022)

Best Master Award 2018/2019 für den besten Masterabschluss in Informatik zwischen 07/2018 und 12/2019 an der Universität zu Lübeck.

Publikationen

Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz (2023). **Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs with Applications**, *Journal of Machine Learning Research (JMLR), Volume 24*.

Malte Luttermann, Marcel Wienöbst, Maciej Liśkiewicz (2023). **Practical Algorithms for Orientations of Partially Directed Graphical Models**, *Proceedings of the Second Conference on Causal Learning and Reasoning (CLeaR 2023)*.

Marcel Wienöbst, Malte Luttermann, Max Bannach, Maciej Liśkiewicz (2023). **Efficient Enumeration of Markov Equivalent DAGs**, *Proceedings of the Thirty-Seventh AAI Conference on Artificial Intelligence (AAAI 2023)*.

Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz (2022). **A New Constructive Criterion for Markov Equivalence of MAGs**, *Proceedings of the Thirthy-Eighth Conference on Uncertainty in Artificial Intelligence (UAI 2022)*. **Best Student Paper**.

Benito van der Zander, Marcel Wienöbst, Markus Bläser, Maciej Liśkiewicz (2022). **Identification in Tree-Shaped Linear Structural Causal Models**, *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence and Statistics (AISTATS 2022)*.

Marcel Wienöbst, Maciej Liśkiewicz (2021). **An Approach to Reduce the Number of Conditional Independence Tests in the PC Algorithm**, *Proceedings of the Forty-Fourth German Conference on AI (KI 2021)*.

Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz (2021). **Extendability of Causal Graphical Models: Algorithms and Computational Complexity**, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021)*. **Best Student Paper**.

Marcel Wienöbst, Max Bannach, Maciej Liśkiewicz (2021). **Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs**, *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*. **Distinguished Paper**.

Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (2020). **PACE Solver Description: PID***, *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*.

Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (2020). **PACE Solver Description: Fluid**, *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*.

Marcel Wienöbst, Maciej Liśkiewicz (2020). **Recovering Causal Structures from Low-Order Conditional Independencies**, *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*.