Counting Background Knowledge Consistent Markov Equivalent Directed Acyclic Graphs

Vidya Sagar Sharma[1]

[1]School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, Maharashtra, India

Abstract

We study the problem of counting the number of directed acyclic graphs in a Markov equivalence class (MEC) that are consistent with *background knowledge* specified in the form of the directions of some additional edges in the MEC. A polynomial-time algorithm for the special case of the problem, when no background knowledge constraints are specified, was given by Wienöbst, Bannach, and Liśkiewicz (AAAI 2021), who also showed that the general case is NP-hard (in fact, #P-hard). In this paper, we show that the problem is nevertheless tractable in an interesting class of instances, by establishing that it is "fixed-parameter tractable": we give an algorithm that runs in time $O(k!k^2n^4)$, where n is the number of nodes in the MEC and k is the maximum number of nodes in any maximal clique of the MEC that participate in the specified background knowledge constraints. In particular, our algorithm runs in polynomial time in the well-studied special case of MECs of bounded tree-width or bounded maximum clique size.

1 INTRODUCTION

A graphical model is a combinatorial tool for expressing dependencies between random variables. Both directed and undirected versions have been used in the literature for modeling different kinds of dependency structures. We study graphical models represented by directed acyclic graphs (DAGs), which represent conditional independence relations and causal influences between random variables by directed edges [Pearl, 2009]. Such graphical models have been used extensively for modeling causal relationships across several fields, e.g., material science [Ren et al., 2020], game theory [Kearns et al., 2001], and biology [Friedman, 2004, Finegold and Drton, 2011].

It is well known that given access only to observational data, the causal DAG underlying a system can only be determined up to its "Markov equivalence class" (MEC) [Verma and Pearl, 1990, Meek, 1995, Chickering, 1995]. Two DAGs are said to be in the same MEC if they model exactly the same set of conditional dependence relations between the underlying random variables. Distinguishing between two DAGs in the same MEC requires the use of interventional data [Hauser and Bühlmann, 2012].

Finding the size of an MEC, therefore, becomes a question of key interest. In particular, the size of an MEC quantifies the uncertainty of the causal model given only observational data. The problem, along with a proposed algorithm, was already mentioned by Meek [Meek, 1995, Section 4.1], and has since then been the focus of a long line of work [Madigan et al., 1996, He et al., 2015, He and Yu, 2016, Bernstein and Tetali, 2017, Ghassami et al., 2019, Talvitie and Koivisto, 2019, Ganian et al., 2020], which culminated in a polynomial time algorithm for the problem by Wienöbst et al. [2021].

Counting with background knowledge constraints In applications, more information about the directions of edges in the underlying DAG than that encoded in the MEC may be available, for example, due to access to domain-specific knowledge. Meek [1995] referred to this as *background knowledge* and modeled it as a specification of the directions of some of the edges of the underlying DAG. Thus, instead of finding the size of the whole MEC, one becomes interested in counting those DAGs in the MEC that are consistent with this specified background knowledge. An algorithm for this problem can also be used as an indicator of the efficacy of a particular intervention in pinning down a DAG within an MEC, by measuring the ratio between the size of the MEC and the number of those DAGs in the MEC that are consistent with the extra background information yielded by the intervention. Wienöbst et al. [2021] showed, however, that in general, this problem is #P-complete (i.e., as hard as counting satisfying assignments to a Boolean for-

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:1911–1920.

mula). Our goal in this paper is to circumvent this hardness result when the specified background knowledge has some special structure.

1.1 OUR CONTRIBUTIONS

We now formalize the above problem. Our input is an MEC on n nodes, and we are given also the directions of another s edges that are not directed in the MEC: we refer to the set of these edges as the *background knowledge*, denoted $\mathcal{K}$. Our goal is to count the number of DAGs in the input MEC that are consistent with the background knowledge $\mathcal{K}$. The above quoted result of Wienöbst et al. [2021] implies that (under the standard P $\neq$ NP assumption) there *cannot* be an algorithm for solving this problem whose run-time is polynomial in both n and s.

Main result The main conceptual contribution of this paper is to define the following parameter which lets us identify important special instances of the problem where we can circumvent the above hardness result. Given the set $\mathcal{K}$ of background knowledge edges, we define the *max-clique-knowledge* k of $\mathcal{K}$ to be the maximum number of vertices in any clique in the input MEC that are part of a background knowledge edge that lies completely inside that clique. In particular, k can be at most twice s, but it can also be much smaller. Our main result (Theorem 3.2) is an algorithm that counts the number of DAGs in the MEC that are consistent with $\mathcal{K}$, and runs in time $O(k! \cdot k^2 \cdot n^4)$.

Discussion and evaluation In particular, the runtime of our algorithm is polynomially bounded when the parameter k above is bounded above by a constant, *even if* the actual size s of the background knowledge is very large. For example, since k is bounded above by the size of the largest clique in the input MEC, it follows that our algorithm runs in polynomial time in the well-studied special case (see, e.g., [Talvitie and Koivisto, 2019]) when the input MEC has constant tree-width (and hence constant maximum-clique size). We provide an empirical exploration of the run-time of our algorithm, including of the phenomenon that it depends on $\mathcal{K}$ only through k and not through s, in Section 6.

1.2 RELATED WORK

It is well known that an MEC can be represented as a partially directed graph, known as an *essential graph*, with special graph theoretic properties [Verma and Pearl, 1990, Meek, 1995, Chickering, 1995, Andersson et al., 1997]. Initial approaches to the problem of computing the size of an MEC by Meek [1995] and Madigan et al. [1996] built upon ideas underlying this characterization. More recently, He et al. [2015] evaluated a heuristic based on the partition of essential graphs into chordal components. Ghassami et al. [2019] gave an algorithm that runs in polynomial time for constant degree graphs, but where the degree of this polynomial *grows* with the maximum degree of the graph.

Our algorithm can be seen as an example of a *fixed parameter tractable* (FPT) algorithm in the well-studied framework of parameterized complexity theory [Cygan et al., 2015]. Parameterized complexity offers an approach to attack computationally hard problems (e.g. those that are NP-hard or #P-hard) by separating the complexity of solving the problem into two pieces – a part that depends purely on the size of the input, and a part that depends only on a well-chosen "parameter" ρ of the problem. An FPT algorithm for a computationally hard problem has a runtime that is bounded above by $f(\rho)\,\mathrm{poly}\,(n)$, where the degree of the polynomial does *not* depend upon the parameter ρ, but where the function f (which does not depend upon the input size n) may potentially be exponentially growing. In our setting, the underlying parameter is the max-clique-knowledge of the background knowledge $\mathcal{K}$.

Thus, the algorithm of Ghassami et al. [2019] cited above is *not* an FPT algorithm. However, Talvitie and Koivisto [2019] improved upon it by giving an FPT algorithm for computing the size of an MEC (without background knowledge): for an undirected essential graph on n nodes whose maximum clique is of size c, their algorithm runs in time $\mathcal{O}(c!2^c c^2 n)$. Finally, Wienöbst et al. [2021] presented the first polynomial time algorithm for computing the size of any MEC (again, without background knowledge). As discussed above, they also showed that counting Markov equivalent DAGs consistent with specified background knowledge is, in general, #P-complete.

We are not aware of any progress towards circumventing this hardness result by imposing specific properties on the specified background knowledge, and to the best of our knowledge, this paper is the first to give a fixed parameter tractable algorithm for the problem. Our algorithm is motivated by the techniques developed by Wienöbst et al. [2021].

2 PRELIMINARIES

We mostly follow the terminology and notation used by Andersson et al. [1997] and Wienöbst et al. [2021] for notions such as ***graph unions***, ***chain graphs***, ***directed*** **and** ***undirected*** **graphs,** ***skeletons***, **v-*structures***, ***cliques, separators, chordal graphs, undirected connected chordal graphs*** (which we will typically denote using the abbreviation **UCCG**), and ***clique trees***. For completeness, we provide detailed definitions in the Supplementary Material.

Notations. A graph G is a pair (V, E), where V is said to be the set of vertices of G, and $E \subseteq V \times V$ is said to be the set of edges of G. For $u, v \in V$, if $(u, v), (v, u) \in E$ then we say there is an undirected edge between u and v, denoted as $u - v$. For $u, v \in V$, if $(u, v) \in E$, and $(v, u) \notin E$ then we say there is a directed edge from u to v, denoted as $u \to v$. For a graph G, we denote V_G as the subset of vertices of G, and E_G as the set of vertices of G. A clique is a set of pairwise adjacent vertices. We denote the set of all maximal cliques of G by $\Pi(G)$. For a set X, we denote by $\#X$ (and sometimes by $|X|$), the size of X.

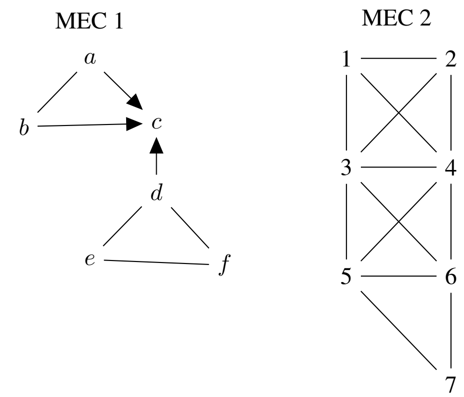
Markov equivalence classes. A causal DAG encodes a set of conditional independence relations between random variables represented by its vertices. Two DAGs are said to belong to the same *Markov equivalence class* (MEC) if both encode the same set of conditional independence relations. Verma and Pearl [1990] showed that two DAGs are in the same MEC if, and only if, both have (i) the same skeleton and (ii) the same set of v-structures. An MEC can be represented by the graph union of all DAGs in it. Andersson et al. [1997] show that a partially directed graph representing an MEC is a chain graph whose undirected connected components (i.e., undirected connected components formed after removing the directed edges) are chordal graphs. We refer to these undirected connected components as *chordal components* of the MEC. With a slight abuse of terminology, we equate the chain graph with chordal components which represents an MEC with the MEC itself, and refer to both as an "MEC".

AMO. Given a partially directed graph G, an *orientation* of G is obtained by assigning a direction to each undirected edge of G. Following Wienöbst et al. [2021], we call an orientation of G an *acyclic moral orientation* (AMO) of G if (i) it does not contain any directed cycles, and (ii) it has the same set of v-structures as G. For an MEC G, we denote the set of AMOs of G by AMO(G).

PEO and LBFS orderings. For an undirected graph, a linear ordering τ of its vertices is said to be a *perfect elimination ordering* (PEO) of the graph if for each vertex v, the neighbors of v that occur after v form a clique. A graph is chordal if, and only if, it has a PEO [Fulkerson and Gross, 1965]. Rose et al. [1976] gave a *lexicographical breadth-first-search* (LBFS) algorithm to find a PEO of a chordal graph (see Algorithm 1 below for a modified version of LBFS). Any ordering of vertices that can be returned by the LBFS algorithm is said to be an *LBFS ordering*.

Representation of an AMO. Given a linear ordering τ of the vertices of a graph G, an AMO of G is said to be *represented* by τ if for every edge $u \to v$ in the AMO, u precedes v in τ. Every LBFS ordering of a UCCG G represents a unique AMO of G, and every AMO of a UCCG G is represented by an LBFS ordering of G (Corollary 1, and Lemma 2 of Wienöbst et al. [2021]). For a maximal clique C of G, we say that C *represents an AMO* α *of* G if there exists an LBFS ordering that starts with C and represents α. Similarly, for a permutation $\pi(C)$ of a maximal clique C of G, we say $\pi(C)$ represents α if there exists an LBFS ordering that starts with $\pi(C)$ and represents α. We denote by AMO$(G, \pi(C))$ and AMO(G, C), the set of AMOs of G that can be represented by $\pi(C)$ and C, respectively.

Canonical representation of AMO. For an AMO α of a UCCG G, Wienöbst et al. [2021] define a unique clique that represents α. We denote the clique as C_α, and say that C_α *canonically* represents α. For the purposes of this paper, we only need certain properties of the canonical representative C_α, and these are quoted in Lemma 4.7. However, for completeness, we also give the definition of C_α in the Supplementary Material.



	Background Knowledge	Max-clique Knowledge
Example 1	$\{a \to b, e \to d, f \to d\}$	3
Example 2	$\{a \to b, e \to d\}$	2
Example 3	$\{1 \to 2, 3 \to 6\}$	2
Example 4	$\{1 \to 2, 2 \to 3, 1 \to 3,$ $3 \to 6, 4 \to 6, 6 \to 7\}$	3

Figure 1: Background Knowledge and Max-clique Knowledge: In MEC 1, there are 2 maximal cliques $\{a, b\}$ and $\{d, e, f\}$. In MEC 2, there are 3 maximal cliques $\{1, 2, 3, 4\}$, $\{3, 4, 5, 6\}$ and $\{5, 6, 7\}$. Examples 1 and 2 are for MEC 1, and examples 3 and 4 are for MEC 2. In example 1, for the maximal clique $\{d, e, f\}$, d and e are part of an edge $e \to d$, and f is part of an edge $f \to d$, i.e., all the nodes of the clique is part of an edge of the background knowledge such that both endpoints of the edge are inside the clique. From the definition of clique knowledge, clique knowledge of the clique $\{d, e, f\}$ is 3. Similarly, clique knowledge of the maximal clique $\{a, b\}$ is 2. This shows the max-clique knowledge of the background knowledge in example 1 for MEC 1 is 3. Similarly, the max-clique knowledge of the background knowledge in example 2 for MEC 1 is 2, and the max-clique knowledge of the background knowledge in examples 3 and 4 for MEC 2 are 2 and 3, respectively.

Background Knowledge and Clique-knowledge. For an MEC represented by a partially directed graph G, *background knowledge*[1] is specified as a set of directed edges

[1]Another way to represent background knowledge is by a

$\mathcal{K} \subseteq E_G$ [Meek, 1995]. A graph G is said to be consistent with $\mathcal{K}$ if, for any edge $u \to v \in \mathcal{K}$, $v \to u \notin E_G$ (i.e., either $u - v \in E_G$ or $u \to v \in G$). For a clique C of G, *clique-knowledge* of $\mathcal{K}$ for C is defined as the number of vertices of C that are part of a background knowledge edge both of whose endpoints are in C. *Max-clique-knowledge* of $\mathcal{K}$ for G is the maximum value of clique-knowledge over all cliques of G.

We denote the set of AMOs of G consistent with background knowledge $\mathcal{K}$ by AMO($G, \mathcal{K}$). Further, for any clique C and any permutation $\pi(C)$ of the vertices of C, we denote by AMO$(G, \pi(C), \mathcal{K})$ and AMO$(G, C, \mathcal{K})$, respectively, the set of $\mathcal{K}$-consistent AMOs of G that can be represented by $\pi(C)$ and C.

3 MAIN RESULT

Andersson et al. [1997] show that a DAG is a member of an MEC G if, and only if, it is an AMO of G. Thus, counting the number of DAGs in the MEC represented by G is equivalent to counting AMOs of G. We start with a formal description of the algorithmic problem we address in this paper.

Problem 3.1 (**Counting AMOs with Background Knowledge**). ***INPUT:*** *(a) An MEC G (in the form of a chain graph with chordal undirected components), (b) Background Knowledge* $\mathcal{K} \subseteq E_G$.

OUTPUT: *Number of DAGs in the MEC G that are consistent with* $\mathcal{K}$*, i.e., #AMO(*$G, \mathcal{K}$*).*

As discussed in the introduction, a polynomial time algorithm for the special case of this problem where $\mathcal{K}$ is empty was given by Wienöbst et al. [2021], in the culmination of a long line of work on that case. However, as discussed in the Introduction, a hardness result proved by Wienöbst et al. [2021] implies, under the standard $P \neq NP$ assumption, that there cannot be an algorithm for Problem 3.1 that runs in time polynomial in both n and $|\mathcal{K}|$.

We, therefore, ask: what are the other interesting cases of the problem which admit an efficient solution? We answer this question with the following result.

Theorem 3.2 (Main result)**.** *There is an algorithm for Problem 3.1 which outputs* $\#AMO(G, \mathcal{K})$ *in time* $O(k!k^2n^4)$*, where* k *is the max-clique-knowledge value of* $\mathcal{K}$ *for G, and* n *is the number of vertices in G.*

In the formalism of parameterized algorithms [Cygan et al., 2015], the above result says that the problem of counting AMOs that are consistent with background knowledge is *fixed parameter tractable*, with the parameter being the max-clique-knowledge of the background knowledge. This contrasts with the #P-hardness result of Wienöbst et al. [2021] for the problem.

The starting point of our algorithm is a standard reduction to the following special case of the problem.

Problem 3.3 (**Counting Background Consistent AMOs in chordal graphs**)**.** ***INPUT:*** *(a) An undirected connected chordal graph (UCCG) G, (b) Background Knowledge* $\mathcal{K} \subseteq E_G$.

OUTPUT: *Number of AMOs of G that are consistent with* $\mathcal{K}$*, i.e., #AMO(*$G, \mathcal{K}$*).*

Proposition 3.4. *Let* G *be an MEC, and let* $\mathcal{K} \subseteq E_G$ *be background knowledge consistent with* G*. Then,* #AMO$(G, \mathcal{K}) = \prod_H$ #AMO$(H, \mathcal{K}[H])$*, where the product ranges over all undirected connected chordal components* H *of the MEC G, and* $\mathcal{K}[H] = \{u \to v : u, v \in H, \text{ and } u \to v \in \mathcal{K}\}$ *is the corresponding background knowledge for* H*.*

Proposition 3.4 reduces Problem 3.1 to Problem 3.3. The proof of Proposition 3.4 follows directly from standard arguments (a similar reduction to chordal components of an MEC has been used in many previous works), and is given in the Supplementary Material.

The rest of the paper is devoted to providing an efficient algorithm for Problem 3.3. Our strategy builds upon the recursive framework developed by Wienöbst et al. [2021]. In the first step, in Section 4, we modify the LBFS algorithm presented by Wienöbst et al. [2021] to make it background aware. This algorithm is used to generate smaller instances of the problem recursively. Further, we modify the recursive algorithm of Wienöbst et al. [2021] to use this new LBFS algorithm. While building upon prior work, both steps require new ideas to take care of the background information. In particular, it is a careful accounting of the background knowledge $\mathcal{K}$ that requires the $k!$ factor in the runtime, where k is the max-clique-knowledge of the background knowledge. In Section 5, we analyze the time complexity of the resulting algorithm. Finally, Section 6 shows our experimental results. Due to space constraints, proofs of many lemmas and theorems are provided in the Supplementary Material. For most of the important results, we provide the proof sketch in the main text.

4 THE ALGORITHM

In this section, we give an FPT algorithm Algorithm 3 that solves Problem 3.3 using a parameter "max-clique knowledge" (Section 2). Algorithm 3 is a background aware version of Algorithm 2 of Wienöbst et al. [2021], which solved the special case of Problem 3.3 when there is no background knowledge. Similar to their algorithm, our algorithm

maximally partially directed acyclic graph (MPDAG). We do not use MPDAGs in our paper because our run time depends only on the number of directed background knowledge edges that generate the MPDAG, and not on the (possibly much larger) number of directed edges in the MPDAG.

uses a modified LBFS algorithm, Algorithm 1, which is a background aware version of the modified LBFS algorithm presented by them. The simple Algorithm 2, which counts background knowledge consistent permutations of a clique, is the new ingredient required in our final algorithm presented in Algorithm 3.

We start with the partitioning of the AMOs. In Section 2, we saw that each AMO of a UCCG is canonically represented by a unique maximal clique of the UCCG. We use this for partitioning the AMOs.

Lemma 4.1. *Let G be a UCCG, and $\mathcal{K}$ be a given background knowledge. Then $\#AMO(G, \mathcal{K})$ equals*

$$\sum_{C \in \Pi(G)} |\{\alpha : \alpha \in AMO(G, \mathcal{K}) \text{ and } C = C_\alpha\}|. \quad (1)$$

Here, $\{\alpha : \alpha \in \text{AMO}(G, \mathcal{K}) \text{ and } C = C_\alpha\}$ is the set of $\mathcal{K}$-consistent AMOs of G that are *canonically* represented by a maximal clique C of G. To compute this, we first compute the union of AMOs of G that are represented by C. The following definition concerns objects from the work of Wienöbst et al. [2021] that are relevant to construct the union graph.

Definition 4.2 ($G^{\pi(C)}, G^C, \mathcal{C}_G(\pi(C)$ and $\mathcal{C}_G(C)$, Wienöbst et al. [2021], Definition 1)**.** *Let G be a UCCG, C a maximal clique of G, and $\pi(C)$ a permutation of C. Then, G^C (respectively, $G^{\pi(C)}$) denotes the union of all the AMOs of G that can be represented by C (respectively, by $\pi(C)$). $\mathcal{C}_G(C)$ (respectively, $\mathcal{C}_G(\pi(C))$) denotes the undirected connected components of $G^C[V_G \setminus C]$ (respectively, $G^{\pi(C)}[V_G \setminus C]$).*

The structure of G^C provides us the set of directed edges in G^C, which helps us to check the $\mathcal{K}$-consistency of G^C. Since G^C is the union of all the AMOs that can be represented by C, a directed edge in G^C is a directed edge in all the AMOs that can be represented by C. Thus, if any such directed edge is not $\mathcal{K}$-consistent then none of the AMOs that can be represented by C can be $\mathcal{K}$-consistent. And, if all the directed edges of G^C are $\mathcal{K}$-consistent then we further reduce our problem into counting $\mathcal{K}$-consistent AMOs for the undirected connected components of G^C (Lemma 4.13).

In order to implement the above discussion, the main insight required is the following definition.

Definition 4.3 ($\mathcal{K}$-consistency of G^C)**.** *Let G be a UCCG and C a maximal clique in G. Given background knowledge $\mathcal{K}$ about the directions of the edges of G, G^C is said to be $\mathcal{K}$-consistent, if there does not exist a directed edge $u \rightarrow v \in G^C$ such that $v \rightarrow u \in \mathcal{K}$.*

As discussed above, Definition 4.3 implies that for a maximal clique C of G, if G^C is not $\mathcal{K}$-consistent then there exists no $\mathcal{K}$-consistent AMO of G that is represented by C. In other words, if G^C is not $\mathcal{K}$-consistent then

$$|\{\alpha : \alpha \in \text{AMO}(G, \mathcal{K}) \text{ and } C = C_\alpha\}| = 0.$$

Then, from Lemma 4.1,

$$\#\text{AMO}(G, \mathcal{K}) = \sum_{C : G^C \text{ is } \mathcal{K}\text{-consistent}} |\{\alpha : \alpha \in \text{AMO}(G, \mathcal{K}) \text{ and } C = C_\alpha\}|.$$

We construct Algorithm 1 to check the $\mathcal{K}$-consistency of G^C, for any maximal clique C of G. Wienöbst et al. [2021] give a modified LBFS algorithm that for input a chordal graph G, and a maximal clique C of G, outputs the undirected connected components (UCCs) of $\mathcal{C}_G(C)$. Their algorithm outputs the UCCs of $\mathcal{C}_G(C)$ in such a way that by knowing the UCCs of $\mathcal{C}_G(C)$, we can construct G^C. We use this fact and construct an LBFS algorithm Algorithm 1, which also checks the $\mathcal{K}$-consistency of G^C. Algorithm 1 is a background aware version of the LBFS algorithm given by Wienöbst et al. [2021].[2]

We now describe our background-aware version of the modified LBFS algorithm: see Algorithm 1. We do not change any line from the LBFS algorithm of Wienöbst et al. [2021]. The modifications we do in their LBFS algorithm are (a) introduction of "flag", at line 2, which is used to check the $\mathcal{K}$-consistency of G^C, (b) lines 11-13, which is used to update the value of "flag", and (c) we also output the value of "flag" with $\mathcal{C}_G(C)$. The correctness of this modification (stated formally in Lemma 4.5) is based on the following observation which in turn uses ideas implicit in the work of Wienöbst et al. [2021].

Observation 4.4 (Implicit in the work of Wienöbst et al. [2021])**.** *Let G be a UCCG, $\mathcal{K}$ be the known background knowledge about G, and C be a maximal clique of G. For input G, C, and $\mathcal{K}$, suppose that at some iteration of Algorithm 1, $\mathcal{L} = \{X_1, X_2, \ldots, X_l\}$. Then,*

1. *For any $u \in C$, and $v \notin C$, if $(u, v) \in E_G$ then $u \rightarrow v$ is a directed edge in G^C.*
2. *For $u \in X_i$, and $v \notin C \cup X_1 \cup X_2 \cup \ldots \cup X_i$, if $(u, v) \in E_G$ then $u \rightarrow v$ is a directed edge in G^C.*
3. *For $u, v \in X_i$, for $1 \leq i \leq l$, if $(u, v) \in E_G$ then $u - v$ is an undirected edge in G^C.*
4. *For $u, v \in C$, $u - v$ is an undirected edge in G^C.*
5. *For any $X_i \in \mathcal{L}$, every undirected connected component of $G[X_i]$ is an element of $\mathcal{C}_G(C)$.*

The following lemma encapsulates the correctness of Algorithm 1.

[2] If Algorithm 1 is executed with $C = \mathcal{K} = \varnothing$, the algorithm performs a normal LBFS with corresponding traversal ordering τ, which is the reverse of a PEO of G.

Algorithm 1: LBFS$(G, C, \mathcal{K})$ (Background aware LBFS, based on the modified LBFS of Wienöbst et al. [2021])

Input : A UCCG G, a maximal clique C of G, and background knowledge $\mathcal{K} \subseteq E_G$.
Output : $(1, \mathcal{C}_G(C))$: if G^C is $\mathcal{K}$-consistent, $(0, \mathcal{C}_G(C))$: otherwise.

```
1  S ← sequence of sets initialized with (C, V \ C)
2  τ ← empty list, L ← empty list, flag ← 1, Y ← empty list
3  while S is non-empty do
4      X ← first non-empty set of S
5      v ← arbitrary vertex from X
6      if v is neither in a set in L nor in C then
7          Append X to the end of the list L.
8          Append undirected connected components of G[X] to the end of Y.
9      end
10     Add vertex v to the end of τ.
11     if u → v ∈ K for any u which is neither in a set in L nor in C then
12         flag = 0;
13     end
14     Replace the set X in the sequence S by the set X \ {v}.
15     N(v) ← {x | x ∉ τ and v − x ∈ E}
16     Denote the current S by (S_1, ..., S_k).
17     Replace each S_i by S_i ∩ N(v), S_i \ N(v).
18     Remove all empty sets from S.
19 end
20 return (flag, Y)
```

Lemma 4.5. *Let G be a UCCG, C be a maximal clique of G, and $\mathcal{K}$ be the known background knowledge about G. For the input G, C, and $\mathcal{K}$, if G^C is not $\mathcal{K}$-consistent Algorithm 1 outputs $(0, \mathcal{C}_G(C))$ on line 20, else it returns $(1, \mathcal{C}_G(C))$ on line 20.*

For any maximal clique C of G such that G^C is $\mathcal{K}$-consistent, to compute the size of the set of $\mathcal{K}$-consistent AMOs of G that are canonically represented by C, we further partition the set based on the different permutations of C. The simple Observation 4.6 below assists us in mapping each AMO of the set to a unique permutation $\pi(C)$ of C.

Observation 4.6. *Let G be a UCCG, and α an AMO of G that is represented by a maximal clique C of G. Then, there exists a unique permutation $\pi(C)$ of C that represents α.*

By slightly extending the definition of the canonical representation of an AMO by a clique, we say that an AMO is **canonically represented by** $\pi(C)$ if the AMO is represented by $\pi(C)$, and also canonically represented by the clique C. Then, Observation 4.6 implies that we can partition the set of $\mathcal{K}$-consistent AMOs that are canonically represented by C into $\mathcal{K}$-consistent AMOs that are canonically represented by its permutations $\pi(C)$, i.e., $\{\alpha : \alpha \in \text{AMO}(G, \pi(C), \mathcal{K}) \text{ and } C = C_\alpha\}$. More formally,

$$|\{\alpha : \alpha \in \text{AMO}(G, \mathcal{K}) \text{ and } C = C_\alpha\}| = \sum_{\pi(C)} |\{\alpha : \alpha \in \text{AMO}(G, \pi(C), \mathcal{K}) \text{ and } C = C_\alpha\}|$$

To compute the size of $\mathcal{K}$-consistent AMOs of G that are canonically represented by $\pi(C)$, we first have to go through the necessary and sufficient conditions for a maximal clique C of G to become C_α, for an AMO α of G.

Lemma 4.7 (Claims 1, 2 and 3 of Wienöbst et al. [2021])**.** *Let G be a UCCG. Wienöbst et al. [2021] fix a rooted clique tree of G to define C_α, for any AMO α of G. Let $\mathcal{T} = (T, R)$ be the rooted clique tree (with root R) of G on which C_α is defined, for each AMO α of G. For an AMO α of G, and a maximal clique C of G, $C = C_\alpha$ if, and only if,*

1. *There exists an LBFS ordering of G that starts with* C*, and represents α, and*
2. *If $\pi(C)$ is the permutation of C that represents α (from Observation 4.6) then there does not exist any edge $C_i - C_j$ in the path in T from R to C such that $\pi(C)$ has a prefix $C_i \cap C_j$.*

The set FP$(C, \mathcal{T})$ is defined to be the set of such forbidden prefixes $C_i \cap C_j$.

Definition 4.8 ($FP(C, \mathcal{T})$, Definition 3 of Wienöbst et al. [2021])**.** *Let G be a UCCG, $\mathcal{T} = (T, R)$ a rooted clique tree of G, C a node in T and $R = C_1 - C_2 - \ldots - C_{p-1} - C_p = C$ the unique path from R to C in T. We define the set* FP$(C, \mathcal{T})$ *to contain all sets of the form $C_i \cap C_{i+1} \subseteq C$, for $1 \leq i < p$.*

Based on Lemma 4.7, we define $(\mathcal{K}, \mathcal{T})$-consistency for a permutation $\pi(C)$ to simplify our computation. This definition is one of the main new ingredients that let us extend the result of Wienöbst et al. [2021].

Definition 4.9 ($(\mathcal{K}, \mathcal{T})$-consistency of permutations of maximal cliques)**.** *Let G be a UCCG, C a maximal clique in G, $\pi(C)$ a permutation of C, $\mathcal{K}$ be a given background knowledge, and $\mathcal{T} = (T, R)$ a rooted clique tree of G (on which C_α is defined). $\pi(C)$ is said to be $(\mathcal{K}, \mathcal{T})$-consistent if (a) $\pi(C)$ is $\mathcal{K}$-consistent, i.e, for any edge $u \to v \in \mathcal{K}$ such that $u, v \in C$, u occurs before v in $\pi(C)$, and (b) no element of FP$(C, \mathcal{T})$ (Definition 4.8) is a prefix of $\pi(C)$.*

If $\pi(C)$ itself is not $\mathcal{K}$-consistent then no $\mathcal{K}$-consistent AMO exists that is represented by $\pi(C)$. Also, if $\pi(C)$ has a prefix in FP$(C, \mathcal{T})$ then from Observation 4.6 and Lemma 4.7, there does not exist an AMO α of G such that α is represented by $\pi(C)$, and $C = C_\alpha$. This yields the following observation.

Observation 4.10. *If $\pi(C)$ is not $(\mathcal{K}, \mathcal{T})$-consistent then there exists no $\mathcal{K}$-consistent AMOs of G that can be canonically represented by $\pi(C)$, i.e., $|\{\alpha : \alpha \in AMO(G, \pi(C), \mathcal{K}) \text{ and } C = C_\alpha\}| = 0$*

We thus focus on only those permutations $\pi(C)$ of C that are $(\mathcal{K}, \mathcal{T})$-consistent. The main ingredient towards this end is the following recursive formula.

Lemma 4.11. *Let G^C be $\mathcal{K}$-consistent. Then, for any $(\mathcal{K}, \mathcal{T})$-consistent permutation $\pi(C)$ of C, the size of the set $\{\alpha : \alpha \in AMO(G, \pi(C), \mathcal{K}) \text{ and } C = C_\alpha\}$ is $\prod_{H \in \mathcal{C}_G(C)} \#AMO(H, \mathcal{K}[H])$.*

Note that the formula obtained in Lemma 4.11 depends only upon the clique C and *not* on the permutation π of the nodes of C (as long as π is itself $\mathcal{K}$-consistent)! This implies immediately that the number of $\mathcal{K}$-consistent AMOs of G for which C is the canonical representative is given by multiplying the product $\prod_{H \in \mathcal{C}_G(C)} \#AMO(H, \mathcal{K}[H])$ with the number of $(\mathcal{K}, \mathcal{T})$-consistent permutation of C.

This motivates us to count $(\mathcal{K}, \mathcal{T})$-consistent permutations of a maximal clique C of G. To count the $(\mathcal{K}, \mathcal{T})$-consistent permutations of C, we define the following:

Definition 4.12. *Let S be a set of vertices, $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$ such that $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_l \subsetneq S$, and $\mathcal{K} \subseteq S \times S$. $\Phi(S, \mathcal{R}, \mathcal{K})$ is the number of $\mathcal{K}$-consistent permutations of S that do not have a prefix in $\mathcal{R}$.*

Lemma 4.1 and Observation 4.10, along with Lemma 4.11 and the discussion following it, finally give us the following recursion.

Lemma 4.13. *Let G be a UCCG, $\mathcal{K}$ be a given background knowledge, and $\mathcal{T} = (T, R)$ a rooted clique tree of G on which $<_\alpha$ has been defined. Then $\#AMO(G, \mathcal{K})$ equals*

$$\sum_C \Phi(C, FP(C, \mathcal{T}), \mathcal{K}[C]) \times \prod_{H \in \mathcal{C}_G(C)} \#AMO(H, \mathcal{K}[H]),$$

where the sum is over those C for which G^C is $\mathcal{K}$-consistent.

Lemma 4.13 solves our counting problem. The precondition of $\Phi(S, \mathcal{R}, \mathcal{K})$ in Definition 4.12 that $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$ has the property $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_l \subsetneq S$ is satisfied at the beginning of the recursion, i.e. when $\mathcal{R} = FP(C, \mathcal{T})$, by Lemma 4.14, which is a consequence of the standard clique intersection property of clique trees of chordal graphs.

Lemma 4.14 (Wienöbst et al. [2021], Lemma 7)**.** *We can order the elements of $FP(C, \mathcal{T})$ as $X_1 \subsetneq X_2 \subsetneq \ldots \subsetneq X_l \subsetneq C$.*

The precondition of $\Phi(S, \mathcal{R}, \mathcal{K})$ is preserved throughout the recursion described in the lemma. We now give a recursive method to compute $\Phi(S, \mathcal{R}, \mathcal{K})$.

Algorithm 2: Valid-Perm$(S, \mathcal{R}, \mathcal{K})$

Input : A clique S, $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$ such that $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_l \subsetneq S$, and background knowledge $\mathcal{K} \subseteq S \times S$.

Output : $\Phi(S, \mathcal{R}, \mathcal{K})$.

```
if R = ∅ then
  | return (|S|!/|V_K|!) · Ψ(V_K, K)
end
sum ← Valid-Perm(S, R − {R_l}, K)
if {(u, v) : u → v ∈ K, v ∈ R_l and u ∉ R_l} ≠ ∅ then
  | return sum
end
return sum − Valid-Perm(R_l, R − {R_l}, K[R_l]) ×
  Valid-Perm(S \ R_l, ∅, K[S \ R_l])
```

Lemma 4.15. *Let S be a clique, and $\mathcal{K} \subseteq S \times S$ be a set of directed edges. Let $R = \{R_1, R_2, \ldots, R_l\}$ where $l \geq 1$ be such that $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_l \subsetneq S$. Then,*

1. *$\Phi(S, \varnothing, \mathcal{K}) = \frac{|S|!}{|V_\mathcal{K}|!} \times \Psi(V_\mathcal{K}, \mathcal{K})$, where $\Psi(V_\mathcal{K}, \mathcal{K})$ is the number of $\mathcal{K}$-consistent permutations of vertices in $V_\mathcal{K}$ ($V_\mathcal{K}$ is the set of end points of edges in $\mathcal{K}$). (Example: Suppose $S = \{1, 2, 3, 4, 5\}$, and $\mathcal{K} = \{1 \to 2, 2 \to 3\}$. Then, $V_\mathcal{K} = \{1, 2, 3\}$. And, there exist only one permutation, $(1, 2, 3)$, of $V_\mathcal{K}$ that is $\mathcal{K}$-consistent, i.e., $\Psi(V_\mathcal{K}, \mathcal{K}) = 1$. This implies $\Phi(S, \varnothing, \mathcal{K}) = 20$.)*
2. *If there exists an edge $u \to v \in \mathcal{K}$ such that $u \in S \setminus R_l$ and $v \in R_l$, then $\Phi(S, R, \mathcal{K}) = \Phi(S, R - \{R_l\}, \mathcal{K})$.*
3. *If there does not exist an edge $u \to v \in \mathcal{K}$ such that $u \in S \setminus R_l$ and $v \in R_l$, then $\Phi(S, R, \mathcal{K}) = \Phi(S, R - \{R_l\}, \mathcal{K}) - \Phi(R_l, R - \{R_l\}, \mathcal{K}[R_l]) \times \Phi(S \setminus R_l, \varnothing, \mathcal{K}[S \setminus R_l])$.*

Proof of Lemma 4.15. Proofs of items 2 and 3 follow easily from the definition of the Φ function (Lemma 4.13), and are similar in spirit to the corresponding results of Wienöbst et al. [2021] in the setting of no background knowledge. We provide the details of these proofs in the supplementary material and focus here on proving item 1. If $\mathcal{R} = \varnothing$ then $\Phi(S, \mathcal{R}, \mathcal{K})$ is the number of $\mathcal{K}$-consistent permutations of S. There are $\frac{|S|!}{|V_\mathcal{K}|!}$ permutations of S consistent with any given ordering of vertices in $V_\mathcal{K}$. The total number of $\mathcal{K}$-consistent permutations of the vertices in $V_\mathcal{K}$ is $\Psi(V_\mathcal{K}, \mathcal{K})$. Therefore, the number of $\mathcal{K}$-consistent permutations of S equals $\frac{|S|!}{|V_\mathcal{K}|!} \times \Psi(V_\mathcal{K}, \mathcal{K})$. □

Algorithm 2 implements Lemma 4.15 to compute $\Phi(S, \mathcal{R}, \mathcal{K})$, and its correctness given below, is an easy consequence of Lemma 4.15.

Algorithm 3: count$(G, \mathcal{K}, \texttt{memo})$ (modification of an algorithm of Wienöbst et al. [2021])

Input : A UCCG G, background knowledge $\mathcal{K} \subseteq E_G$.
Output : $\#\text{AMO}(G, \mathcal{K})$.

```
1  if G ∈ memo then
2  |  return memo[G]
3  end
4  𝒯 = (T, R) ← a rooted clique tree of G
5  if R = V_G then
6  |  memo[G] = Φ(V, ∅, 𝒦)
7  |  return memo[G]
8  end
9  sum ← 0
10 Q ← queue with single element R
11 while Q is not empty do
12 |  C ← pop(Q)
13 |  push(Q, children(C))
14 |  (flag, ℒ) ← LBFS(G, C, 𝒦)
15 |  if flag = 1 then
16 |  |  prod ← 1
17 |  |  foreach H ∈ ℒ do
18 |  |  |  prod =
   |  |  |    prod × count(G[H], 𝒦[H], memo)
19 |  |  end
20 |  |  sum = sum + prod × Φ(C, FP(C, 𝒯), 𝒦[C])
21 |  end
22 end
23 memo[G] = sum
24 return sum
```

Observation 4.16. *For input $S, \mathcal{R} = \{R_1, R_2, \ldots, R_l\}$, and $\mathcal{K}$, where $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_l \subsetneq S$, and $\mathcal{K} \subseteq S \times S$, Algorithm 2 returns $\Phi(S, \mathcal{R}, \mathcal{K})$.*

We now construct Algorithm 3 that computes $\#\text{AMO}(G, \mathcal{K})$. Algorithm 3 evaluates this formula, utilizing memoization to avoid recomputations.

Theorem 4.17. *For a UCCG G and background knowledge $\mathcal{K}$, Algorithm 3 returns $\#AMO(G, \mathcal{K})$.*

Proof of Theorem 4.17: We first fix a clique tree $\mathcal{T} = (T, R)$ (at line 4) on which we define $<_\alpha$. Lines 5-8 deals with the base case when G is a clique. If G is not a clique, Algorithm 3 follows Lemma 4.13. The full detail is given in Supplementary Material due to lack of space.

5 TIME COMPLEXITY ANALYSIS

In this section, we analyze the run time of Algorithm 3. The proof of the following observation, which shows that despite our modifications, Algorithm 1 still runs in linear time, is given in the Supplementary Material.

Observation 5.1. *For a UCCG G, a maximal clique C of G, and background knowledge $\mathcal{K}$, Algorithm 1 runs in linear time $O(|V_G| + |E_G|)$.*

Similar to Wienöbst et al.'s `count` function, our `count` function (Algorithm 3) is also recursively called at most $2|\Pi(G)| - 1|$ times, where $\Pi(G)$ is the set of maximal cliques of G. Our approach to compute the background aware version of Φ (Algorithm 2) is similar to that of Wienöbst et al. [2021], and the difference in time complexity comes from the high time complexity of computation of $\Phi(S, \varnothing, \mathcal{K})$ at item 1 (it is $O(1)$ for $\mathcal{K} = \varnothing$, which is the setting considered by Wienöbst et al. [2021]). Proof of the claims below can be found in the Supplementary Material.

Proposition 5.2. *Let G be a UCCG, and $\mathcal{K}$ be the known background knowledge about G. The number of distinct UCCG explored by the `count` function (as defined in Algorithm 3) is bounded by $2|\Pi(G)| - 1$.*

Lemma 5.3. *For input S, $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$, and $\mathcal{K}$, Algorithm 2 can be implemented using memoization to use $O(k! \cdot k^2 \cdot |\Pi(G)|^2)$ arithmetic operations, where k is the max-clique knowledge of $\mathcal{K}$ (assuming factorials of integers from 1 to $|V_G|$ are available for free).*

Theorem 5.4 (Final runtime bound of Algorithm 3)**.** *For a UCCG G, and background knowledge $\mathcal{K}$, Algorithm 3 runs in time $O(k!k^2n^4)$, more precisely $O(k!k^2 \cdot |\Pi(G)|^4)$, where n is the number of nodes in G, and k is the max-clique knowledge of $\mathcal{K}$.*

Proof of Theorem 3.2. Together, Theorems 4.17 and 5.4 prove our main result, Theorem 3.2. □

6 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of Algorithm 3 on a synthetic dataset. For each $n \in \{500, 510, 520, \ldots, 1000\}$, we construct 50 random chordal graphs with n nodes, and for each $k \in \{5, 6, \ldots, 13\}$, we construct a set of background knowledge edges with k as its max clique knowledge value. We then measure the running time of Algorithm 3 for each of these (graph, background knowledge) pairs, and take the mean running time over all such pairs with the same value of n and k. Further details about the construction of these instances can be found in the Supplementary Material.

Validating the run-time bound To validate the $O(k!k^2n^4)$ run-time bound established in Section 5, we draw log-log plots of the mean run-time T against the size n of the graph, for each fixed value of k (fig. 2). As predicted by the polynomial (in n) run-time bound in our theoretical result, we get, for each value of the parameter k, a roughly linear log-log plot.

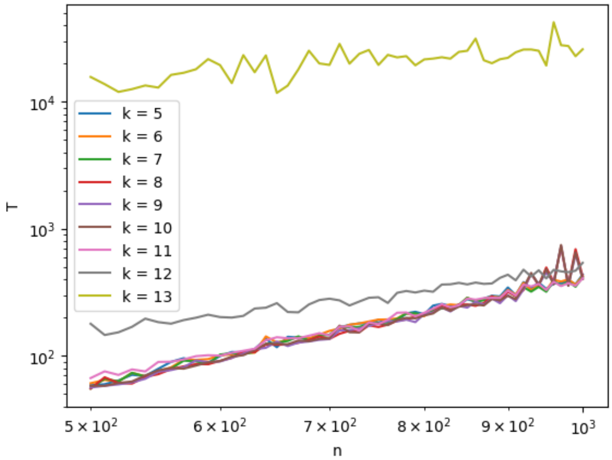


Figure 2: log vs. log plot of T vs. n

The intercept of the log-log plot While the plots in fig. 2 for $k \in \{5, 6, 7, 8, 9, 10, 11\}$ are quite close to each other, the separation of the plots for $k = 12$ and $k = 13$ is much larger. The reason behind it is the actual time complexity of Algorithm 3 (explained in Supplementary Material) can roughly be bounded as $\log T \leq \log a + \log(k!k^2 + b) + 4 \log n$, where a and b are constants independent of n and k. The above observation then shows that until about $k \approxeq 11$, all the plots have intercepts close to each other, as the value of b dominates $k!k^2$ for small value of k. The difference starts increasing fast when the $k!k^2$ term becomes larger than b.

Effect of the size of the background knowledge An important feature of our analysis of Algorithm 3 is that its run-time bound does not depend directly upon the actual size of the background knowledge. To validate this, we conduct the following experiment: we fix a chordal graph of size n and the max-clique knowledge value k, and then construct two different sets $\mathcal{K}_1$ and $\mathcal{K}_2$ of background knowledge edges, of different sizes such that both have the same k value (the details of the construction are given in the Supplementary Material). In table 1, T_1, T_2 are the running times of Algorithm 3 with background knowledge $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively. The table confirms the expectation that when the

n	k	$\lvert\mathcal{K}_1\rvert$	$\lvert\mathcal{K}_2\rvert$	T_1	T_2
1000	10	65	116	370	353
1000	10	75	137	346	338
1100	11	55	121	467	455
1100	11	51	104	460	453

Table 1: Exploring runtime dependence on the number of background knowledge edges

graph and k are fixed, the running time does not increase much when the size of the background-knowledge increases. More detailed data and discussion of this phenomenon are given in the Supplementary Material.

7 CONCLUSION

Our main result shows that the max-clique-knowledge parameter we introduce plays an important role in the algorithmic complexity of counting Markov equivalent DAGs under background knowledge constraint. In particular, it leads to a polynomial time algorithm in the special case of graphs of bounded maximum-clique size. Note that an algorithm that runs in polynomial time in the general case is precluded by the #P-hardness result of Wienöbst et al. [2021] (unless P = NP). However, the optimal dependence of the run time on the max-clique-knowledge parameter is an interesting open problem left open by our work.

8 ACKNOWLEDGMENT

We acknowledge the support from the Department of Atomic Energy, Government of India, under project no. RTI4001. We want to thank Piyush Srivastava for his invaluable suggestions, discussions, and help in the completion of this paper. We thank all the anonymous reviewers for multiple useful suggestions which helped in improving the presentation of this paper.

References

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

Megan Bernstein and Prasad Tetali. On Sampling Graphical Markov Models. *arXiv:1705.09717*, 2017.

David Maxwell Chickering. A Transformational Characterization of Equivalent Bayesian Network Structures. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI 1995)*, pages 87–98, 1995.

Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

Michael Finegold and Mathias Drton. Robust Graphical Modeling of Gene Networks using Classical and Alternative t-Distributions. *Annals of Applied Statistics*, pages 1057–1080, 2011.

Nir Friedman. Inferring Cellular Networks using Probabilistic Graphical Models. *Science*, 303(5659):799–805, 2004.

Delbert Fulkerson and Oliver Gross. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics*, 15: 835–855, 1965.

Robert Ganian, Thekla Hamm, and Topi Talvitie. An Efficient Algorithm for Counting Markov Equivalent DAGs. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, volume 34, pages 10136–10143, 2020.

AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Kun Zhang. Counting and Sampling from Markov Equivalent DAGs using Clique Trees. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, volume 33, pages 3664–3671, 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.

Yangbo He and Bin Yu. Formulas for Counting the Sizes of Markov Equivalence Classes of Directed Acyclic Graphs. *arXiv:1610.07921*, 2016.

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(1): 2589–2609, 2015.

Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical Models for Game Theory. In *Proceedings of the 17th Conference on Uncertainity in Artifical Intelligence conference (UAI 2001)*, pages 253–260, 2001.

David Madigan, Steen A. Andersson, Michael D. Perlman, and Chris T. Volinsky. Bayesian Model Averaging and Model Selection for Markov Equivalence Classes of Acyclic Digraphs. *Communications in Statistics–Theory and Methods*, 25:2493–2519, 1996.

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

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2009.

Zekun Ren, Felipe Oviedo, Maung Thway, Siyu I. P. Tian, Yue Wang, Hansong Xue, Jose Dario Perea, Mariya Layurova, Thomas Heumueller, Erik Birgersson, Armin G. Aberle, Christoph J. Brabec, Rolf Stangl, Qianxiao Li, Shijing Sun, Fen Lin, Ian Marius Peters, and Tonio Buonassisi. Embedding Physics Domain Knowledge into a Bayesian Network Enables Layer-by-Layer Process Innovation for Photovoltaics. *NPJ Computational Materials*, 6(1):9, January 2020.

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

Topi Talvitie and Mikko Koivisto. Counting and Sampling Markov Equivalent Directed Acyclic Graphs. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, volume 33, pages 7984–7991, 2019.

Thomas Verma and Judea Pearl. Equivalence and Synthesis of Causal Models. In *Proceedings of the 6th Conference on Uncertainity in Artifical Intelligence conference (UAI 1990)*, pages 220–227, 1990.

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)*, pages 12198–12206, 2021.