Efficient Enumeration of Markov Equivalent DAGs

Marcel Wienöbst[1], **Malte Luttermann**[2], **Max Bannach**[1], **Maciej Liśkiewicz**[1]

[1] Institute for Theoretical Computer Science, University of Lübeck, Germany
[2] Institute of Information Systems, University of Lübeck, Germany
{wienoebst@tcs, luttermann@ifis, bannach@tcs, liskiewi@tcs}.uni-luebeck.de

Abstract

Enumerating the directed acyclic graphs (DAGs) of a Markov equivalence class (MEC) is an important primitive in causal analysis. The central resource from the perspective of computational complexity is the delay, that is, the time an algorithm that lists all members of the class requires between two consecutive outputs. Commonly used algorithms for this task utilize the rules proposed by Meek (1995) or the transformational characterization by Chickering (1995), both resulting in superlinear delay. In this paper, we present the first linear-time delay algorithm. On the theoretical side, we show that our algorithm can be generalized to enumerate DAGs represented by models that incorporate background knowledge, such as MPDAGs; on the practical side, we provide an efficient implementation and evaluate it in a series of experiments. Complementary to the linear-time delay algorithm, we also provide intriguing insights into Markov equivalence itself: All members of an MEC can be enumerated such that two successive DAGs have structural Hamming distance at most three.

1 Introduction

Graphical causal models endow researchers with an intuitive and mathematically sound language to infer causal relations between random variables from observational and interventional data. Directed acyclic graphs (DAGs), whose edges encode direct causal influences between the variables, belong to the most popular models and are used in many areas of empirical research (Spirtes, Glymour, and Scheines 2000; Rothman et al. 2008; Pearl 2009; Koller and Friedman 2009; Elwert 2013). However, there is usually not a unique DAG that can be learned from observational or limited experimental data as multiple models can encode the same statistical properties. These DAGs form a *Markov equivalence class* (MEC) and each of them explains the data equally well (Andersson, Madigan, and Perlman 1997; Pearl 2009).

Exploring the structural and quantitative properties of MECs are challenging tasks in graphical causal analysis and, despite extensive research efforts, several basic issues involving MECs fundamental to causal discovery remain open as e. g., calculating the number of MECs on n variables (Gillispie and Lemieux 2001; Steinsky 2003) or enumerating them efficiently (Chen, Choi, and Darwiche 2016).

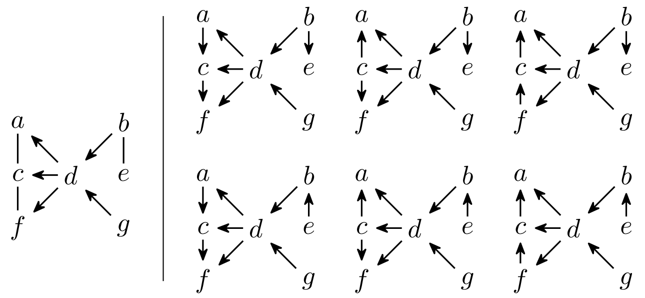

Figure 1: A Markov equivalence class (MEC) on the right, which consists of six DAGs. This class is represented by the left CPDAG, which uniquely represents the MEC by including undirected edges if two DAGs differ in their direction.

In this work, we study the properties of a *single* MEC, encoded as a *completed partially directed acyclic graph* (CPDAG) (Andersson, Madigan, and Perlman 1997), see Fig. 1 for an example. The CPDAG representation is often learned directly by causal discovery algorithms (Spirtes, Glymour, and Scheines 2000; Chickering 2002) and, thus, it is of high practical value to offer efficient implementations for their analysis. Recently, Wienöbst, Bannach, and Liśkiewicz (2021b) have shown that computing the size of an MEC as well as uniformly sampling from it can be done in polynomial time. We deal with the closely related problem of *enumerating the DAGs in an MEC* given its CPDAG, that is listing each member of the MEC exactly once. 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) and `causaldag` (Squires 2018). 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, Kalisch, and Bühlmann 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 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 work.

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 the *delay:* the algorithm's run-time between two consecutive output DAGs. Another desirable property would be that the 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 the present work, 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 utilizes the rules proposed by Meek (1995) to transform a causal graph (e. g., a CDPAG or PDAG) into its maximal orientation. Applying these rules has the property that any remaining 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. Hence, the MEC can be explored through such edge reversals starting from an arbitrary DAG in the class. The issue with this approach 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 Appendix A as well as a rigorous analysis of their delay.

The main contribution of this paper is the first $O(n + m)$-delay algorithm that, for a given CPDAG representing an MEC, lists all members of the class.[1] 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 the 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 our ideas can be used in the more general setting of enumerating maximal ancestral graphs (MAGs) which encode conditional independence relations in DAG models with latent variables (Richardson and Spirtes 2002).

[1]We denote the number of vertices by n and the number of edges by m.

2 Preliminaries

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq V \times V$. An edge $u - v$ is undirected if $(u, v), (v, u) \in E$ and directed $u \to v$ if $(u, v) \in E$ and $(v, u) \notin E$. Vertices linked by an edge (of any type) are *adjacent* and *neighbors* of each other. We say that u is a *parent* of v if $u \to v$. We denote by $Pa(v)$ and $Ne(v)$ the set of parents and neighbors of v. The *degree* $\delta(v)$ of vertex v is the number of its neighbors $|Ne(v)|$. The structural Hamming distance (SHD) of $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ is denoted by $\mathrm{shd}(G_1, G_2)$ and defined as number of pairs $(a, b) \in V^2$ with differing edge relations, i. e., $E_1 \cap \{(a, b), (b, a)\} \neq E_2 \cap \{(a, b), (b, a)\}$. Given a graph $G = (V, E)$ and a vertex set S, the *induced subgraph* $G[S]$ contains the edges $E \cap (S \times S)$ from G that are incident only to vertices in S. The union of a set of graphs $\{G_1 = (V, E_1), \ldots, G_k = (V, E_k)\}$ is the graph $G = (V, \bigcup_{i=1}^{k} E_k)$. A path π between two vertices v_1 and v_p is a sequence of distinct vertices $\pi = \langle v_1, \ldots, v_p \rangle$ with $p \geq 2$ such that each vertex v_i is adjacent to v_{i+1} for $i = 1, \ldots, p - 1$. An *undirected connected component* is a maximal induced subgraph in which every pair of vertices is connected by a path of undirected edges. A path of the form $v_1 \to v_2 \to \ldots \to v_p$ is directed or *causal*. A graph is *acyclic* if there is no directed path from a vertex u to v with $v \to u$. An acyclic graph with only directed edges is called a DAG. An undirected graph is called *chordal* if no subset of four or more vertices induces an undirected cycle.

The *skeleton* of G, denoted by $\mathrm{skel}(G)$, is a graph with the same vertex set in which every edge is replaced by an undirected edge. A *v-structure* is an ordered triple of vertices (u, c, v) that induces the subgraph $u \to c \leftarrow v$. A Markov equivalence class (MEC) consists of DAGs encoding the same set of conditional independence relations among the variables. (Verma and Pearl 1990; Frydenberg 1990) showed that two DAGs are Markov equivalent iff they have the same skeleton and the same v-structures. An MEC can be represented by a CPDAG (*completed partially directed acyclic graph*), which is the union graph of the DAGs in the equivalence class it represents. The set $[G]$ denotes all DAGs in the MEC represented by CPDAG G.

Subclasses of MECs can be represented by *partially directed acyclic graphs* (PDAGs), which are restricted only in that they may not contain a directed cycle, and *maximally oriented PDAGs* (MPDAGs), which consist of PDAGs closed under the four *Meek rules*. Explicitly, we only use the first

Meek rule in this work, which states that an induced subgraph $a \to b - c$ is oriented into $a \to b \to c$. A formal description of all four Meek rules can be found in Appendix A.1.

The input of the enumeration algorithms is the representation of an MEC in form of its CPDAG G. Hence, before approaching the enumeration task, it is important, first and foremost, to understand how one can derive a DAG in $[G]$ from the CPDAG representation (this is also called the *extension task* and such a DAG is called a *consistent extension* of G). To extend G into a consistent DAG, it is necessary to find an orientation of its undirected edges. This orientation needs to be acyclic and contain the same v-structures as G.

Fact 1 (Andersson, Madigan, and Perlman (1997); He, Jia, and Yu (2015)). *The undirected components of a CPDAGs are* undirected and connected chordal graphs *(UCCGs). Acyclically orienting each UCCG independently, without introducing a v-structure, gives a DAG in* $[G]$.

The acyclic orientations without a v-structure of a chordal graph are called *AMOs* (acyclic moral orientations). *Every* DAG in $[G]$ may be computed by finding appropriate AMOs for the UCCGs. In Fig. 1, to obtain the DAG at the top left, the UCCG $a - c - f$ is oriented as AMO $a \to c \to f$ and $b - e$ is oriented as $b \to e$. For the former there are three AMOs, for the latter two, corresponding to the six DAGs in the MEC (as the UCCGs can be oriented independently). Thus, when tackling the enumeration task for CPDAGs, it suffices to enumerate the AMOs of a chordal graph.

Fact 2 (Implicit in Wienöbst, Bannach, and Liśkiewicz (2021b)). *Every AMO of a UCCG G can be obtained by orienting the edges according to an MCS ordering ($a \to b$ if a comes before b in the ordering).*

An MCS ordering is a linear ordering of the vertices produced by running the graph traversal algorithm *Maximum Cardinality Search* (MCS) (Tarjan and Yannakakis 1984), which was originally proposed for testing chordality of a graph in linear time. Notably, the reverse direction holds as well, that is, an MCS ordering will always produce an AMO of G. A brief introduction to chordal graphs, AMOs and the MCS algorithm is given in Appendix A.3.

The output of an MCS depends on the choices of the "next" vertex to visit in each step of the graph traversal. This vertex is taken from the set of vertices with the largest number of already visited neighbors (called the vertices with *highest label*) and, from Fact 2, we can conclude that there are such choices, which may produce any AMO of a given UCCG.

3 Enumerating AMOs with Linear Delay

The observations from the previous section yield a new approach: Instead of producing a single AMO by choosing an arbitrary vertex in each step of the MCS, we perform multiple choices and recur for each of them – eventually listing all AMOs. There is one pitfall however: We *cannot* simply choose *every* vertex from the highest-label set one after the other as some graphs would be output multiple times. Fig. 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.

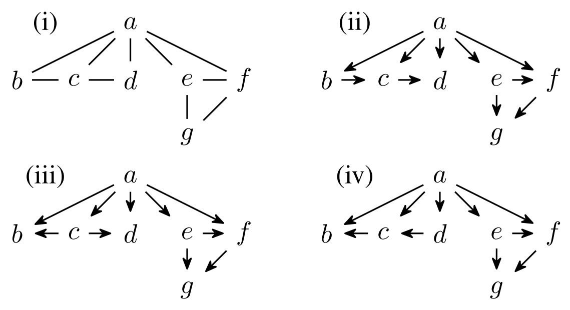

Figure 2: An example showing for the chordal graph (i) that, at a given step of the algorithm, not all vertices in the highest-label set can be chosen. If the MCS starts with vertex a, all neighbors b, c, d, e, f have the same label, namely 1. While an MCS may choose any one of them, we *cannot* choose *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 highest-label vertices 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).

This issue can be addressed as follows: While the choice of the first vertex v from the highest-label set is arbitrary, every other vertex x has to be connected to v in the remaining graph (the induced subgraph over the unvisited vertices). 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 with highest label 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 1. *Given a chordal graph $G = (V, E)$ and the sequence of previously visited vertices τ with $|\tau| = k < n$ produced by an MCS with current highest-label set S.*

1. *If $x, y \in S$ are connected in $G[V \setminus \tau]$, the set of AMOs produced by choosing x next is disjoint from the set produced by choosing y next.*
2. *If $x, y \in S$ are unconnected in $G[V \setminus \tau]$, any AMO produced by choosing y as the next vertex can be produced by choosing a vertex in S connected to x in $G[V \setminus \tau]$ next.*

Proof. We show item 1 first and let p be the shortest path between x and y in $G[V \setminus \tau]$. Since AMOs do not contain v-structures, any AMO choosing x before y must orient p as $x \to \cdots \to y$ while any other AMO yields $x \leftarrow \cdots \leftarrow y$.

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

The following Lemma provides a simplified way of testing whether two vertices of highest label are connected.

Algorithm 1: Linear-time delay algorithm MCS-ENUM for listing all AMOs of a chordal graph.

```
   input  : A UCCG $G = (V, E)$.
   output : All AMOs of $G$.
 1 $A :=$ array of $n$ initially empty sets
 2 $\tau :=$ empty list
 3 $A[0] := V$
 4 enumerate$(G, A, \tau)$
 5 function enumerate ($G$, $A$, $\tau$)
 6     if $|\tau| = n$ then
 7         Output AMO of $G$ according to ordering $\tau$
 8     end
 9     $i :=$ highest index of non-empty set in $A$
10     $v :=$ any vertex from $A[i]$
11     $x := v$
12     do
13         delete $x$ from $A[i]$
14         append $x$ to $\tau$
15         foreach $w \in (Ne(x) \setminus \tau)$ do
16             $j :=$ index of set in $A$ that $w$ is in
17             delete $w$ from $A[j]$
18             insert $w$ in $A[j+1]$
19         end
20         enumerate ($G$, $A$, $\tau$)
21         foreach $w \in (Ne(x) \setminus \tau)$ do
22             $j :=$ index of set in $A$ that $w$ is in
23             delete $w$ from $A[j]$
24             insert $w$ in $A[j-1]$
25         end
26         insert $x$ in $A[i]$
27         pop $x$ from $\tau$
28         if $x = v$ then
29             $R := \{a \mid a$ reachable from $v$ in $G[A[i]]\}$
30         end
31     while $R$ is non-empty, $x := pop(R)$
32 end
```

Lemma 2. *Given a connected chordal graph $G = (V, E)$ and a sequence of visited vertices τ produced by an MCS with the current highest-label set S. Vertices $x, y \in S$ are connected in $G[S]$ iff they are connected in $G[V \setminus \tau]$.*

Algorithm MCS-ENUM utilizes these results to enumerate the AMOs of a chordal graph. Lines 1 to 4 in Algorithm 1 initialize the necessary data structures for an MCS (in particular an array A, which includes the set of vertices with i visited neighbors at index $A[i]$). Afterward, the recursive function enumerate is called. It first chooses any vertex v from the vertices with most visited neighbors (just as a normal MCS). This vertex is then removed from $A[i]$ in line 13, it is appended to τ in line 14, which stores the traversal sequence (later used to impose edge directions based on its ordering) and the neighbors are moved from $A[j]$ to $A[j+1]$ in lines 15 to 19. The recursive calls to enumerate are repeated until, at some point, the first AMO is output in line 7 (up to this point there is no difference to an MCS).

After the recursive call, however, the changes are reversed in lines 21 to 27 and then, in line 29, the vertices reachable from v are computed. These are then iterated in the do-while loop, meaning we also recursively go through the AMOs produced by choosing those vertices instead of v, as discussed above. Note that reachability in line 29 is only performed once in a call of enumerate for the initial vertex v.

Theorem 1. *Given a chordal graph G,* MCS-ENUM *enumerates all AMOs of G.*

Proof. Every DAG output in line 7 is an AMO, as it is generated by a linear ordering produced by an MCS. This holds as any chosen vertex is from the highest-index non-empty set in A. To see that the algorithm outputs *all* AMOs of G, recall that every AMO can be represented by an MCS ordering by Fact 2. In principle, Algorithm 1 considers all possible courses an MCS could take, except the pruning of vertices unreachable from v. By Lemma 2, it suffices to inspect only connected vertices in $G[A[i]]$ and by item 2 of Lemma 1 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 1 it follows that τ_1 and τ_2 yield different AMOs. □

Theorem 2. MCS-ENUM *has worst-case delay $O(n + m)$.*

Proof. 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), lines 21 to 31 are executed and the do-while loop stops (otherwise we would be in phase (ii)). The for-loop in lines 21 to 25 has time complexity $O(\delta(x))$. Moreover, the reachability query executed in line 29 does not yield any vertices (otherwise the do-while loop would continue), meaning it takes time $O(\delta(x))$ (all neighbors of v are checked once and the search stops). The run-time is therefore $O(m)$ as every edge is considered at most twice.

The main costs of phase (iii) are produced by the for-loop in lines 15 to 19, which requires time $O(\delta(x))$ leading to an overall time of $O(m)$. Both for-loops are executed in phase (ii), resulting in overall time $O(\delta(x))$. The reachability query from v in line 29 costs time $O(m)$. As this is done only once, we obtain a worst-case delay of $O(n + m)$. □

The result immediately generalizes to CPDAGs.

Theorem 3. *The Markov equivalence class $[G]$ of a CPDAG G can be enumerated with worst-case delay $O(n + m)$.*

Proof. Algorithm 1 also works for unconnected chordal graphs without any modifications. Hence, it can be 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 Fact 1. □

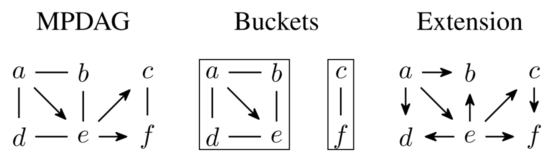

Figure 3: The MPDAG on the left, contains two buckets with chordal skeleton (middle), which lead to a consistent extension when each oriented into an AMO (right).

4 PDAGs and MPDAGs

One can naturally generalize the enumeration task to only consider members of the MEC, which conform to certain background knowledge given in the form of additional directed edges. Such a subclass of an MEC is commonly represented by an MPDAG (or PDAG). As before, a DAG is a consistent extension of G if it has the same skeleton and v-structures.It is easy to see that MEEK-ENUM works in this generalized setting as well, as does CHICKERING-ENUM (see also Corollary 2 in Section 5). In this section, we show how MCS-ENUM can be adapted, first to handle MPDAGs and, building on this, PDAGs.

Definition 1. *We term the graph induced by the vertices of a undirected component in an MPDAG a* bucket.

Buckets are similar to the undirected chordal components that we considered in the previous section, in that orienting each bucket in an MPDAG without cycles or v-structures will yield a consistent extension. However, a bucket may already contain directed edges. An illustration is given in Fig. 3.

Fact 3 (Wienöbst, Bannach, and Liśkiewicz (2021a))**.** *An MPDAG is extendable (i. e., has a consistent extension) iff the skeleton of every bucket is chordal. An extension can be computed in time $O(n+m)$.*

An MPDAG can be extended by running a modified MCS for each bucket B in the following way: The graph traversal is performed on the skeleton of B with the restriction that only vertices in the highest-label set S that have no unvisited parent in B are considered. These are the vertices x with $Pa(x) \setminus \tau = \emptyset$, given that τ contains the visited vertices, and we denote the set of such vertices by $S^+ \subseteq S$. This way, the MCS conforms to the background edges.

With these insights, an analogue modification of Algorithm 1 suggests itself to enumerate *all* AMOs of a bucket: Perform the algorithm on the skeleton of the bucket and only consider vertices in S^+.

Lemma 3. *Let B be a bucket and τ be a sequence of visited vertices with $|\tau| = k < n$ produced by the modified MCS using S^+. Then it holds that:*

1. *If $x, y \in S^+$ are connected in skel$(B[V \setminus \tau])$, the set of AMOs produced by choosing x next is disjoint from the set produced by choosing y next.*
2. *If $x, y \in S^+$ are unconnected in skel$(B[V \setminus \tau])$, any AMO produced by choosing y as the next vertex can also be produced by choosing a vertex in S^+ connected to x in skel$(B[V \setminus \tau])$ 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 1 applies). Then x or y have an incoming edge due to the non-applicability of the first Meek rule in the original bucket B. This violates the assumption that $x, y \in S^+$, meaning that τ_1 and τ_2 imply different AMOs. For item 2 the same argument as in Lemma 1 holds. □

Reachability can again be tested in a simplified way:

Lemma 4. *Given a bucket B and a sequence of visited vertices τ produced by the modified MCS using S^+. Vertices $x, y \in S^+$ are connected in $B[V \setminus \tau]$ iff they are connected in $B[S^+]$.*

Using these results, we can show that:

Theorem 4. *There is an algorithm that enumerates all AMOs of a given bucket B with worst-case delay $O(n+m)$.*

Proof sketch. Consider the just sketched algorithm, i. e., which proceeds as Algorithm 1 for the skeleton of B with the modification of choosing vertices and performing reachability with regard to S^+ (the algorithm is given explicitly in Appendix B.2). By using S^+, the resulting AMOs conform with the directed edges in the bucket and due to Lemma 3 and 4 and by similar arguments as for Theorem 1 every such AMO is output exactly once. The linear-time delay follows as before, notably, S^+ can be efficiently maintained by storing the in-degree of each vertex. □

Using Fact 3, the result for buckets immediately generalizes to MPDAGs.

Corollary 1. *There is an algorithm that enumerates all consistent extensions of a given MPDAG with linear-time delay.*

The matter for PDAGs is similar as they can be maximally oriented into an equivalent MPDAG by Meek's rules.

Theorem 5. *There is an algorithm that enumerates all consistent extensions of a given PDAG with linear-time 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 (Wienöbst, Bannach, and Liśkiewicz 2021a). Afterward, apply the algorithm from Corollary 1. □

Wienöbst, Bannach, and Liśkiewicz (2021a) showed that the initialization step of maximally orienting a PDAG is likely not possible in linear time. However, using a finer complexity analysis, it can be performed in time $O(dm)$, where d is the *degeneracy* of the input's skeleton, which implies linear-time on many natural graph classes such as planar graphs, bounded-degree and bounded-treewidth graphs.

5 Another Approach for Enumerating Markov Equivalent DAGs

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

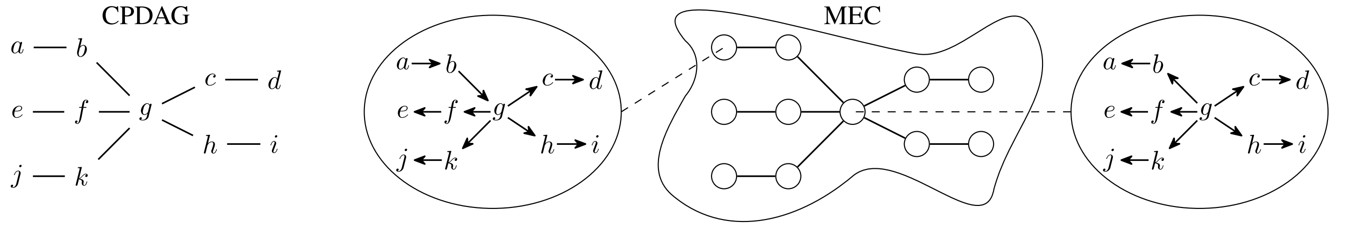

Figure 4: An example that has no sequence of SHD two that enumerates all Markov equivalent DAGs. Two DAGs in the MEC are connected by an edge if they can be transformed into each other by a single edge reversal. For trees, the resulting topology coincides with the one of the CPDAG, each DAG in the MEC 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.

enumeration sequence of small changes between consecutive DAGs. While having a worse delay, such sequences are more natural from the causal perspective, with only a few edge orientations changing at a time, and provide structural insights into Markov equivalence itself. 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 following characterization of Markov equivalence, which is also the basis for CHICKERING-ENUM (see also Appendix A.2):

Fact 4 (Chickering (1995)). *For any two Markov equivalent DAGs D and D' there is a sequence of Markov equivalent DAGs $\langle D = D_1, \ldots, D_k = D' \rangle$ such that D_i and D_{i+1} have SHD one.*

The statement also holds for two consistent extensions of a *PDAG* in the sense that all intermediate DAGs are also consistent extensions of this PDAG.

Corollary 2. *For any two consistent extensions C and C' of PDAG G there is a sequence of consistent extensions of G $\langle C = D_1, \ldots, D_k = C' \rangle$ such that D_i and D_{i+1} have SHD one.*

Proof. Only differing edges between C and C' are reversed in the constructive proof of Chickering (1995). Hence, all background edges stay fixed during the transformation. □

This means that it is possible to go from one DAG to another with only single edge reversals for CPDAGs as well as for PDAGs and MPDAGs. The task we are trying to solve, however, is to enumerate *all* members of an MEC, 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. Fig. 4 provides an example that does not even allow a sequence of SHD *two*.

However, if we permit *three* edge reversals between consecutive DAGs we can always find such a sequence:

Theorem 6. *Every MEC can be represented as sequence $\langle D_1, D_2, \ldots \rangle$ of Markov equivalent DAGs such that D_i and D_{i+1} have SHD at most three.*

Proof. For a constructive, proof consider the graph that contains all DAGs in the MEC as nodes.[2] 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 Fact 4 the graph is connected.

Every connected graph has a sequence $\langle p_1, p_2, \ldots \rangle$ that contains every node exactly once such that the distance between consecutive nodes is at most three.[3] 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. □

Due to Corollary 2, this result generalizes to PDAGs:

Corollary 3. *The consistent extensions of PDAG G can be represented as sequence $\langle D_1, D_2, \ldots \rangle$ of consistent extensions of G with SHD at most three.*

We note that MEEK-ENUM, CHICKERING-ENUM and MCS-ENUM do not have this property.

Lemma 5. *Sequences of DAGs produced by* MEEK-ENUM *may contain consecutive DAGs with SHD larger than three.*

Proof. Consider the CPDAG shown in Fig. 4. Since MEEK-ENUM has no preferences on the edge it orients first, it may start with the edge $a \to b$. All other edge directions would then follow from the first Meek rule yielding the output DAG shown in Fig. 4. The orientation $a \leftarrow b$ is tried afterward,

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

[3]The authors became aware of this graph property due to a problem posed by Jorke de Vlas in the annual programming contest BAPC (Problem H at BAPC 2021: https://2021.bapc.eu/).

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 4 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 afterward choose i as the first vertex – yielding again the same DAGs.

Corollary 4. *Sequences of DAGs produced by* CHICKERING-ENUM *and* MCS-ENUM *may contain consecutive DAGs with structural Hamming distance larger than three.*

Computationally, our results do not imply a better bound on the delay in producing the sequence from Theorem 6 and we leave this as an open problem. The constructive algorithm (which we call SHD3-ENUM) given in the proof of Theorem 6 behaves 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 (between two outputs a constant number of recursive calls are handled; see Appendix B.3). 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 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 brief introduction to MAGs and a more detailed analysis are given in Appendix C.

Corollary 5. *Every MEC of MAGs without selection bias can be represented as sequence $\langle M_1, M_2, \dots \rangle$ of Markov equivalent MAGs with SHD at most three.*

In contrast, approaches such as MEEK-ENUM do not exist for MAGs as analogue rules proposed by Zhang (2008) only complete the graph in case the edge marks are inferred by 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.

6 Experiments

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.

In Fig. 5, we compare the average delay of the four approaches (MEEK-ENUM, CHICKERING-ENUM, MCS-ENUM, SHD3-ENUM) implemented in Julia (Bezanson et al. 2017). For each instance, the programs were terminated after two minutes if the enumeration was not completed. As the enumeration problem reduces to listing the AMOs of a chordal graph (as shown in Fact 1), we consider as instances undirected graphs generated by randomly inserting edges, which do not violate chordality, until a graph with $3 \cdot n$ edges is reached. Note that these instances are all CPDAGs, just fully undirected ones, and thus all approaches can be applied to this setting. In Appendix D, we also compare the results for

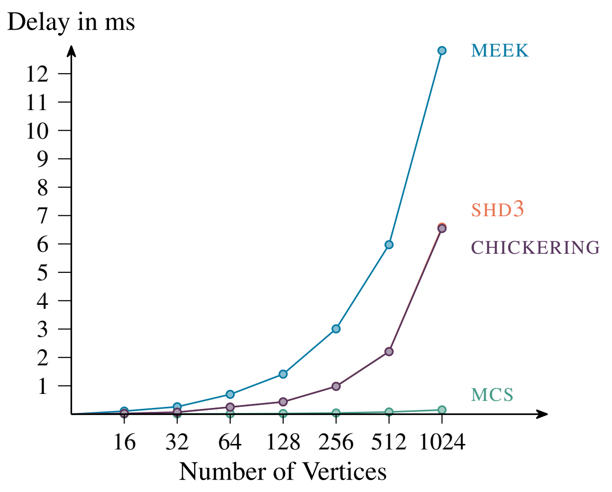

Figure 5: Average delay in milliseconds for enumerating the AMOs of random chordal graphs with $m = 3 \cdot n$ edges. We compare the algorithms MEEK-ENUM, CHICKERING-ENUM, MCS-ENUM and SHD3-ENUM.

CPDAGs *with* directed edges as well as for PDAGs, which both lead to very similar results. Moreover, we discuss the distribution of the delay for the various algorithms.[4]

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 four 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 (showing that the enumeration with SHD at most three gives mainly structural insights into Markov equivalence and has in itself no computational advantage).

7 Conclusion

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 so general that they 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 cannot be applied, because similar structure does not exist for Markov equivalence of MAGs or, at least, is not known.

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

Acknowledgments

The research of Malte Luttermann is partly supported by the Medical Cause and Effects Analysis (MCEA) project.

References

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

Bezanson, J.; Edelman, A.; Karpinski, S.; and Shah, V. B. 2017. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1): 65–98.

Chen, E. Y.-J.; Choi, A. C.; and Darwiche, A. 2016. Enumerating equivalence classes of Bayesian networks using EC graphs. In *Artificial Intelligence and Statistics*, 591–599. PMLR.

Chickering, D. M. 1995. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, UAI '95*, 87–98.

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

Elwert, F. 2013. Graphical Causal Models. In *Handbook of Causal Analysis for Social Research*, Handbooks of Sociology and Social Research, 245–273. Springer.

Frydenberg, M. 1990. The chain graph Markov property. *Scandinavian Journal of Statistics*, 333–353.

Gillispie, S. B.; and Lemieux, C. 2001. Enumerating Markov Equivalence Classes of Acyclic Digraph Models. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, UAI '01*, 171–177.

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

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

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

Kalisch, M.; Mächler, M.; Colombo, D.; Maathuis, M. H.; and Bühlmann, P. 2012. Causal inference using graphical models with the R package pcalg. *Journal of statistical software*, 47: 1–26.

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

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

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

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

Richardson, T.; and Spirtes, P. 2002. Ancestral graph Markov models. *The Annals of Statistics*, 30(4): 962–1030.

Rothman, K. J.; Greenland, S.; Lash, T. L.; et al. 2008. *Modern epidemiology*, volume 3. Wolters Kluwer Health/Lippincott Williams & Wilkins Philadelphia.

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

Squires, C. 2018. *`causaldag`: creation, manipulation, and learning of causal models*.

Steinsky, B. 2003. Enumeration of labelled chain graphs and labelled essential directed acyclic graphs. *Discrete mathematics*, 270(1-3): 267–278.

Tarjan, R. E.; and Yannakakis, M. 1984. 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.

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

Wienöbst, M.; Bannach, M.; and Liśkiewicz, M. 2021a. Extendability of Causal Graphical Models: Algorithms and Computational Complexity. In *Proceedings of the 37th Conference in Uncertainty in Artificial Intelligence, UAI '21*. AUAI Press.

Wienöbst, M.; Bannach, M.; and Liśkiewicz, M. 2021b. Polynomial-Time Algorithms for Counting and Sampling Markov Equivalent DAGs. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2021*, volume 35, 12198–12206. AAAI Press.

Zhang, J. 2008. 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.

Zhang, J.; and Spirtes, P. 2005. A transformational characterization of Markov equivalence for directed acyclic graphs with latent variables. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI '05*, 667–674.