Appendix – 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

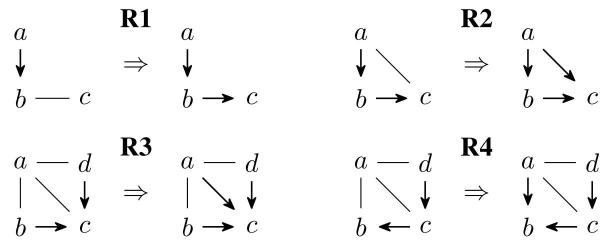

Figure 6: The four Meek rules that are used to characterize MPDAGs (Meek 1995).

A Formal Description of Related Algorithms

A.1 Enumerating Markov Equivalent DAGs Based on Meek's Rules

Meek (1995) gave a complete set of four rules (R1 - R4), which, when applied repeatedly, transform a PDAG into its maximal orientation, i. e., orient all undirected edges, which are fixed in the DAGs represented by the PDAG. Graphs completed under the Meek rules are also called *Maximally oriented PDAGs* (MPDAGs).

See R1 in Figure 6 as an example for the application of the rules. The edge $b - c$ necessarily has a fixed orientation because $a \rightarrow b \leftarrow c$ and $a \rightarrow b \rightarrow c$ are in different MECs (the first graph contains a v-structure, the latter does not) and hence cannot be represented by the same PDAG. The single DAG represented by the PDAG in this case is $a \rightarrow b \rightarrow c$, the graph which does not introduce a new v-structure, and it is immediately obtained after applying R1. In most cases some undirected edges remain even after application of the Meek rules.

In the converse, a graph completed under the Meek rules has the property that every undirected edge $x - y$ is *not* fixed in the DAGs it represents, i. e., there is a DAG in the class which contains $x \rightarrow y$ and one which contains $x \leftarrow y$. Thus, the members can be enumerated by first orienting the edge as $x \rightarrow y$ (and recursively orienting the remaining edges one-by-one, always after completing the graph under the Meek rules to ensure that every orientation yields a valid DAG) and afterward orienting the edge as $x \leftarrow y$. This approach we call MEEK-ENUM is depicted in Algorithm 2.

```
input  : A CPDAG $G = (V, E)$.
output : $[G]$.
function enumerate($G$)
    $H :=$ maximal orientation of $G$
    undir $:= \{\{u, v\} \mid (u, v) \in E_H \wedge (v, u) \in E_H\}$
    if undir is empty then
        Output $H$
    else
        $\{u, v\} :=$ any element from undir
        $E_H := E_H \setminus \{(u, v)\}$   // Orient $u \leftarrow v$
        enumerate($H$)
        $E_H := E_H \cup \{(u, v)\}$   // Undo
        $E_H := E_H \setminus \{(v, u)\}$   // Orient $u \rightarrow v$
        enumerate($H$)
        $E_H := E_H \cup \{(v, u)\}$   // Undo
    end
end
```

Algorithm 2: Enumeration algorithm of Markov equivalent DAGs based on Meek's rules (MEEK-ENUM).

MEEK-ENUM has polynomial-delay, particularly as every orientation leads to a valid DAG (there are no "dead-ends" in the recursive search). However, it requires the application of Meek's rules in every step, which amounts to significant effort to perform between two consecutive outputs. Naively estimated, checking whether one of the four Meek rules applies takes time $O(n^4)$ and in a worst-case scenario this check has to be performed multiple times as more and more edges get oriented successively. This would lead to a polynomial time requirement of large degree, even for applying the Meek rules once. However, there have been recent algorithmic improvements in this regard.

Fact 5 (Wienöbst, Bannach, and Liśkiewicz (2021))**.** *Given a PDAG G, Meek's rules can be applied exhaustively on G in time $O(n^3)$.*

This gives a slightly better bound on the achievable delay, formally stated in the following Theorem.

Theorem 7. MEEK-ENUM *can be implemented such that the MEC is enumerated with worst-case delay $O(m \cdot n^3)$.*

Proof. Clearly, every DAG in the MEC is output as every undirected edge is oriented in both directions (by soundness of the Meek rules every directed edge is actually fixed in the DAGs in the class). No DAG is output twice because in each step, an undirected edge is oriented and fixed, i.e., every recursive call receives a different graph as input. Nor is an invalid DAG output, because every undirected edge can be oriented in both directions leading to a valid DAG by completeness of the Meek rules. In the worst case, the algorithm needs m recursive calls until a DAG is output, and a single call costs time $O(n^3)$ due to the application of Meek's rules by Fact 5. Hence, the delay between two outputs is bounded by $O(m \cdot n^3)$. □

The same result holds for PDAGs and MPDAGs without modifications. We note that in the experiments, we decided to implement the Meek rules in the "naive" way as this is typically used in implementations, which makes for a more meaningful comparison.

A.2 Enumerating Markov Equivalent DAGs Based on Chickering's Transformational Characterization

Another approach for enumerating Markov equivalent DAGs is build on the characterization by Chickering (1995), also stated briefly in Section 5 above. It is based on the notion of *covered edges*:

Definition 2 (Chickering (1995)). *An edge $x \to y$ is called* covered *if* $\mathrm{Pa}(x) \cup \{x\} = \mathrm{Pa}(y)$.

This allows the following characterization of Markov equivalence:

Theorem 8 (Chickering (1995)). *Let D and D' be two Markov equivalent DAGs. There exists a sequence of $\mathrm{shd}(D, D')$ edge reversals in D with the following properties:*

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

Hence, starting from a DAG D we reach all DAGs in the same Markov equivalence class by reversals of covered edges. This permits the following enumeration approach, which we call CHICKERING-ENUM.

Starting from D, similar to a depth-first-search (DFS), all neighbors of D (graphs with a single reversed covered edge) are explored and this is continued recursively. Eventually all Markov equivalent DAGs are reached by Theorem 8 above. In order to not visit any DAG twice, it is necessary to store a set of all visited DAGs.

Theorem 9. CHICKERING-ENUM *enumerates a Markov equivalence class and can be implemented with worst-case delay $O(m^3)$.*

Proof. Clearly, any DAG is output exactly once by Theorem 8 and storing all visited DAGs ensures not outputting any DAG multiple times.

For the delay, it is first useful to analyze when the most steps between two outputs are necessary. This happens when

```
input  : A CPDAG $G = (V, E)$.
output : $[G]$.
1  $D :=$ any DAG in $[G]$
2  vis $:= \{D\}$
3  enumerate($D$)
4  function enumerate($D$, vis)
5      Output $D$
6      foreach DAG $D'$ obtained by reversing a covered edge in $D$ do
7          if $D' \notin$ vis then
8              vis $:=$ vis $\cup \{D'\}$
9              enumerate($D'$, vis)
10         end
11     end
12 end
```

Algorithm 3: Enumeration algorithm of Markov equivalent DAGs based on Chickerings' characterization (Chickering 1995) (CHICKERING-ENUM).

the algorithm traverses back up the recursion tree (from a leaf), potentially up to the root, without outputting any new DAG and then down into another subtree. In principle, the recursion depth might be exponential and this would, in turn, lead to an exponential-time delay.

However, it is possible to bound the recursion depth by m (by Theorem 8 every edge is reversed at most once) and one will still find all DAGs in the MEC. It remains to estimate the cost at each recursion step. Going up the recursion tree, at each DAG, $O(m)$ neighbors might be checked whether they have been visited before, e. g., if they are in vis, and all of them might indeed are. The check takes time $\Omega(m)$ per neighboring DAG D' as the whole DAG has to be "read" at least once for lookup. If one uses a hash table for storing the DAGs, expected time $O(m)$ can be reached.

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

A.3 Computing a Consistent Extension of a CPDAG based on the MCS Algorithm

In this subsection, we revisit the closely related problem of finding an arbitrary DAG in an MEC represented by a CPDAG G instead of listing all of them. This is known as the *extension* task and it is well-known that it is possible to perform it in linear-time (Hauser and Bühlmann 2012) for CPDAGs using the so-called Lexicographic Breadth-First Search (LBFS) algorithm, which is a graph traversal algorithm originally proposed for testing chordality of a graph. For MPDAGs and PDAGs, this problem is discussed in depth in (Wienöbst, Bannach, and Liśkiewicz 2021).

As we base our results on the Maximum Cardinality Search (MCS) algorithm, which has been used less frequently in the causality community compared to LBFS, we give a brief introduction. The MCS algorithm has been designed, as LBFS, for testing chordality of a graph. It is a graph traversal algorithm, which visits the vertices of an undirected graph in an order representing[1] an AMO (*acyclic moral orientation*, that is, an orientation without directed cycles and v-structures) iff the graph is chordal.[2]

The connection between AMOs and CPDAGs stems from the fact that the undirected components of a CPDAG are chordal and replacing each by an AMO leads to a DAG in the Markov equivalence class. The reason for this is that a CPDAG is extended into a DAG by orienting its undirected edges without creating a directed cycle or a new v-structure. This is why AMOs are needed, and the undirected components are chordal, because only chordal graphs have AMOs.

```
input  : A CPDAG $G = (V, E)$.
output : DAG $D \in [G]$.
1  $D :=$ copy of $G$
2  $U := (V, \{(u, v) \mid (u, v)$ and $(v, u) \in E\})$, i. e., the
     graph with only the undirected edges of $G$
3  foreach connected component $C = (V_C, E_C)$ of $U$ do
4      $\tau_C = \texttt{mcs}(C)$
5      Orient edges in $D[V_C]$ according to $\tau_C$, i. e.,
         $x - y$ as $x \to y$ if $x$ comes before $y$ in $\tau_C$
6  end
7  function mcs ($C$)
8      $A :=$ array of $n$ initially empty sets
9      $\tau :=$ empty list
10     $A[0] := V$
11     while $|\tau| < |V_C|$ do
12         $i :=$ highest index of non-empty set in $A$
13         $v :=$ any vertex from $A[i]$
14         delete $v$ from $A[i]$
15         append $v$ to $\tau$
16         foreach $w \in (Ne(v) \setminus \tau)$ do
17             $j :=$ index of set in $A$ that $w$ is in
18             delete $w$ from $A[j]$
19             insert $w$ in $A[j + 1]$
20         end
21     end
22     return $\tau$
23 end
```

Algorithm 4: Computing a DAG in the MEC represented by CPDAG G in linear-time $O(n + m)$.

We can now look at Algorithm 4, which implements this strategy, in more detail. We discussed the first part (lines 2 to 6) in detail. The UCCGs (i.e., the connected components when all directed edges are removed from G) are oriented one-by-one. For this, an ordering τ is computed according to which the edges are oriented. This ordering τ is the traversal sequence of an MCS. This algorithm is given in lines 7 to 23. It differs from standard graph traversals in the way the next vertex to visit is chosen (line 13). The algorithm chooses one of the vertices with most previously visited neighbors. This is implemented efficiently by having a set of vertices with 0 previously visited neighbors ($A[0]$), a set for vertices with 1 previously visited neighbor ($A[1]$), and so on. When a vertex v is handled (lines 14 to 20), its neighbors are moved to a higher set (line 19).

This can be done in constant time per neighbor (Tarjan and Yannakakis 1984), meaning the MCS runs in linear-time. As it is performed on disjoint subgraphs, we obtain overall linear-time $O(n + m)$ for computing a DAG in $[G]$.

Theorem 10. *Algorithm 4 computes a DAG in the MEC represented by CPDAG G in time $O(n + m)$.*

B Missing Proofs and Algorithms

B.1 Section 3

We begin by giving the proof of Lemma 2:

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 $G[V \setminus \tau]$.*

Proof. Trivially, if x and y are connected in $G[S]$ so they are in $G[V \setminus \tau]$. Hence, we only have to prove that if they are not connected in $G[S]$, then they are also disconnected in $G[V \setminus \tau]$. The special case that the highest label has label 0 is trivial as then we have $S = V \setminus \tau$. So let τ be non-empty and consider two arbitrary vertices $x, y \in S$ that are in different connected components of $G[S]$. For a contradiction assume that they are connected in $G[V \setminus \tau]$ via a shortest path $\pi = x - p_1 - \cdots - p_\ell - y$ with $\ell \geq 1$ and $p_1, \ldots, p_\ell \in V \setminus \tau$. Note that π is an unchorded path (i. e., one where only successive vertices are connected by an edge) due to being the shortest path between x and y. Also denote the visited neighbors of vertex v by $P(v) = \tau \cap Ne(v)$.

In π, there has to be a vertex $p_i \notin S$, else x and y are connected in $G[S]$. Hence, there is (i) a vertex $z \in P(x)$, which is not in $P(p_i)$, and (ii) a vertex $z' \in P(y)$, which is not in $P(p_i)$, because x and y have higher label than p_i, meaning $|P(x)| > |P(p_i)|$ and $|P(y)| > |P(p_i)|$.

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

The remaining case is that z and z' are both not a common neighbor of x and y. We show that there can be no AMO produced by the MCS, when it chooses x or y next from S, which would be a contradiction. W.l.o.g., we show the argument for x. Let α be a topological ordering inducing an AMO of G having prefix $\tau + x$. Denote by $\alpha^{-1}(v)$ the

[1]A linear ordering of the vertices represents the orientation of an undirected graph, in which edges are directed $x \to y$ if x comes before y in the ordering.

[2]In the chordal graph literature the term *perfect elimination ordering* is more commonly used in this setting. A linear-ordering of the vertices describes an AMO iff its reverse is a perfect elimination ordering. We will stick to AMOs in this paper to avoid confusion.

position of v in α. It has to hold that $\alpha^{-1}(x) < \alpha^{-1}(p_1) < \alpha^{-1}(p_2) < \cdots < \alpha^{-1}(p_\ell) < \alpha^{-1}(y)$ as the first vertex p_j with $\alpha^{-1}(p_j) > \alpha^{-1}(p_{j+1})$, would induce a v-structure. As we have $\alpha^{-1}(z') < \alpha^{-1}(p_\ell) < \alpha^{-1}(y)$, vertices z' and p_ℓ have to be adjacent to avoid a v-structure. Then, the same applies to $\alpha^{-1}(z') < \alpha^{-1}(p_{\ell-1}) < \alpha^{-1}(p_\ell)$, which implies an edge between z' and $p_{\ell-1}$. This iteration can be continued only until vertex p_i, which is nonadjacent to z' by assumption. □

For the sake of completeness, we also explicitly state the EnumMCS algorithm for a CPDAG G. The enumeration task immediately reduces to finding the AMOs of the chordal components of G.

```
input  : A CPDAG $G = (V, E)$.
output : $[G]$.
1 $U := (V, \{(u, v) \mid (u, v)$ and $(v, u) \in E\})$, i. e., the
    graph with only the undirected edges of $G$
2 Execute Algorithm 1 on $U$ and each time add the
    directed edges of $G$ when outputting the DAG (in
    line 7 of Algorithm 1).
```

Algorithm 5: Linear-time delay enumeration algorithm of Markov equivalent DAGs based on EnumMCS.

B.2 Section 4

We give the proof of Lemma 4 omitted in the main paper:

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 $B[S^+]$.*

Proof. Recall that S is the set of *all* highest-label vertices including the ones with unvisited parents. As shown in the proof of Lemma 2 the vertices x and y are connected in S by the chordality of B. For a contradiction assume that x and y are connected in S but not in S^+. Then there is a shortest path $x = p_1 - p_2 - \cdots - p_{k-1} - p_k = y$ with some $p_i \in S \setminus S^+$. Consider the p_i with smallest i and assume the shortest path is chosen such that this i is maximized (in case that there are multiple shortest paths). This path is completely undirected as x or y would otherwise have an incoming edge by the same argument as in the proof of Lemma 3.

Observe that p_i has an incoming edge from an unvisited vertex z_1. By the properties of the MCS we have $z_1 \in S$ as otherwise z_1 would not be connected to a previous neighbor a of p_i implying the v-structure $a \rightarrow p_i \leftarrow c$ and violating the fact that the modified MCS returns an AMO. Therefore, z_1 needs to be connected to p_{i-1} and p_{i+1}. Furthermore, if z_1 has an incoming edge from an unvisited vertex z_2 then this vertex has to be connected to p_{i-1} and p_{i+1} as well.

This process can be iterated further and we will consider the first z_j that has no unvisited parent. Such a vertex has to exist as otherwise there would be a directed cycle. The edge between z_j and p_{i-1} is undirected by the minimality of p_i.

The edge between z_j and p_{i+1} may be directed. But if it is, there has to be an edge $z_j - p_{i+2}$. Then for some p_l there is an undirected edge $z_j - p_l$ as the edge $z_j - y$ has to be undirected. This yields the desired contradiction as either the path $x = p_1 - \cdots - p_{i-1} - z_j - p_l - \ldots p_k = y$ is shorter than the previously considered one or the first vertex with an unvisited parent has a higher index. □

We explicitly state the generalization of EnumMCS to buckets, which was only sketched in the main paper. The differences to Algorithm 1 are highlighted.

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

Algorithm 6: Linear-time delay enumeration algorithm of consistent extensions of a bucket.

Theorem 11. *There is an algorithm that enumerates all AMOs of a given bucket with worst-case delay $O(n + m)$.*

Proof. This algorithm is given in Algorithm 6. We first show that every DAG that is output by the algorithm is an AMO. This follows from the fact that the output is produced by an

MCS. Notice that the restriction to vertices with no unvisited parents leads to no violation of this fact as the algorithm still *only* chooses vertices with the highest-label. That such a vertex always exists was shown in (Wienöbst, Bannach, and Liśkiewicz 2021).

The next thing we prove is that every AMO of the given bucket is output. By Fact 2 every AMO can be represented by an MCS ordering. Which particular AMO is produced by an MCS depends on the specific choices of highest-label vertices. Our algorithm considers all possible choices at each step except (i) the ones where a vertex has unvisited parents and (ii) the ones where the vertex is unreachable from the first chosen vertex v in a recursion step.

Clearly, (i) only excludes AMOs incompatible with the given directed edges (background knowledge). For (ii) observe that if a vertex is unreachable from v, it does not matter whether it is chosen before or after v by item 2 of Lemma 3.

Finally, it remains to show that no AMO is output twice. Clearly, every sequence τ that we obtain is different. So assume for the sake of contradiction that the algorithm outputs two sequences τ_1 and τ_2 that represent the same AMO. Let x and y be the vertices in τ_1 and τ_2 at the first differing position. Then x and y are connected in the induced subgraph over the unvisited vertices (by line 29) in the skeleton of B. A contradiction since this implies that the AMOs represented by τ_1 and τ_2 differ.

For the complexity analysis we partition the algorithm into three phases (i), (ii), and (iii) as in the proof of Theorem 2. The cost of (ii) and (iii) are the same as in the analysis of Theorem 2. For (i) just observe that reachability runs in $O(d)$ due to Lemma 4. □

We also explicitly state the algorithm for enumerating PDAGs (Algorithm7). The algorithm also works for MPDAGs, in which case the highlighted line 4 does not need to be executed. The algorithm computes the buckets and runs Algorithm 6 on them.

```
input  : A PDAG $G = (V, E)$.
output : All consistent extensions of $G$.
1 if G has no consistent extensions then
2     return $\emptyset$
3 end
4 $G :=$ the maximal orientation of $G$
    (i. e., its completion under the Meek rules)
5 $D :=$ copy of $G$
6 $B := (V, \{(u, v) \mid (u, v) \in E$
    and $u, v$ are connected by undirected edges in $G\})$,
    i. e., the graph containing only the buckets of $G$
7 Execute Algorithm 6 on $B$ and each time add the
    remaining edges of $G$ when outputting the DAG (in
    line 8 of Algorithm 6).
```
Algorithm 7: Linear-time delay enumeration algorithm of the consistent extensions of PDAG G based on MCS-ENUM for buckets (Algorithm 6).

B.3 Section 5

We complement the structural results in Section 5 in the main paper, by explicitly stating the algorithm (called SHD3-ENUM) described textually in the proof of Theorem 6 and analyzing its delay. The differences to CHICKERING-ENUM (Algorithm 3) are highlighted.

```
input  : A CPDAG $G = (V, E)$.
output : $[G]$.
1 $D :=$ any DAG in $[G]$
2 vis $:= \{D\}$
3 enumerate$(D, \text{vis}, 0)$
4 function enumerate ($D$, vis, $i$)
5     if $i \bmod 2 = 0$ then
6         Output $D$
7     end
8     foreach DAG $D'$ obtained by reversing a covered
        edge in $D$ do
9         if $D' \notin$ vis then
10            vis $:=$ vis $\cup \{D'\}$
11            enumerate ($D'$, vis)
12        end
13    end
14    if $i \bmod 2 = 1$ then
15        Output $D$
16    end
17 end
```
Algorithm 8: Enumeration algorithm with SHD ≤ 3 of Markov equivalent DAGs based on Chickerings' characterization (Chickering 1995) (SHD3-ENUM).

The main paper uses this algorithm to show that it is possible to enumerate an MEC with successive DAGs having SHD at most three. Here, we study the algorithmic properties of the algorithm.

Theorem 12. *For a CPDAG G,* SHD3-ENUM *enumerates the DAGs in $[G]$ with delay $O(m^2)$.*

Proof. The argument is similar to the proof of Theorem 9. First, it is clear that all DAGs in the MEC are output exactly once.

The analysis of the delay differs slightly in the sense that this algorithm has worst-case delay $O(m^2)$ (instead of $O(m^3)$) due to the fact that between two outputs only a constant number of recursive calls are handled. This follows from the fact that the algorithm outputs successive DAGs with SHD ≤ 3 and, hence, between the output of these DAGs, there are only constantly many steps in the traversal of the MEC. The cost per DAG can be bounded by $O(m^2)$ as in Theorem 9. □

Both Algorithm 8 as well as Theorem 12 directly apply to PDAGs as well.

C MAGs – Causal Graphs under Latent Confounding

MAGs without selection bias are mixed graphs $G = (V, E)$ with two types of edges: directed $x \to y$ and bidirected $x \leftrightarrow y$. The semantics are different compared to DAGs in that edges encode ancestral relations, i. e., a directed edge means that x is an ancestor (a cause, but not necessarily a direct cause) of y and a bidirected edge means that neither x is a cause of y nor is y a cause of x (as simplification, think that there is a latent confounder between x and y). An excellent introduction to MAGs is given in (Zhang 2008).

Zhang and Spirtes (2005) gave the following transformational characterization of Markov equivalent MAGs without selection bias (also called *DMAGs*):

Theorem 13 (Zhang and Spirtes (2005))**.** *Two DMAGs G and G' are Markov equivalent iff there exists a sequence of single edge mark changes in G such that*

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

It follows that the graph over Markov equivalent DMAGs with an edge between DMAGs with SHD 1 (we define the SHD similar to the DAG case, that is, the number of differing edges; one could also define it to be the number of differing edge marks, and the statement still holds) is connected and hence, there is, by the same argument as in Theorem 6, a sequence of DMAGs containing each in the MEC exactly once with successive DMAGs having SHD ≤ 3, which proves Corollary 5.

We consider it out of the scope of this work to investigate the algorithmic aspects of enumerating Markov equivalent MAGs in detail, but deem this an fascinating topic for further research. In particular, it would be highly interesting to develop enumeration algorithms based on the other strategies discussed in this paper.

Finally, we note that the transformational characterization in Theorem 13 does not hold for MAGs *with* selection bias and it follows from the example given in (Zhang and Spirtes 2005) that it is indeed not possible to find an enumeration sequence with SHD ≤ 3 in this case.

D Further Experimental Results

This section shows the full experimental results of the four algorithms (MEEK-ENUM, CHICKERING-ENUM, MCS-ENUM, SHD3-ENUM) with results for CPDAGs and PDAGs, as well as for slightly denser graphs. Fig. 7 depicts the results. The instances are generated as follows:

- For the undirected chordal graphs, a random tree was generated, to which randomly drawn edges are inserted, which do not violate chordality. This is done until the graph has $k \cdot n$ edges. In the main paper, we considered $k = 3$, here we choose slightly denser graphs with $k = \log_2 n$.
- The CPDAGs are generated by first creating a DAG D randomly, whose CPDAG representation is computed afterward. The DAG is, on the one hand, sampled analogously to the undirected case, by adding directed edges, which do not violate acyclicity until $k \cdot n$ edges are reached. On the other hand, we generate DAGs based on scale-free graphs (using the Barabási-Albert model (Albert and Barabási 2002)), which are oriented by imposing a random topological ordering.
- The PDAGs are generated by first sampling CPDAGs (in the way described above) and then orienting further edges (imitating the addition of background knowledge). For this, between three and seven undirected edges (this number is chosen uniformly at random) are oriented in the CPDAG, with the Meek rules being applied after each orientation.

For CPDAGs and PDAGs, as for the chordal graphs, we choose the parameters $k = 3$ and $k = \log_2 n$. The four algorithms were run on each of the instances for a maximum of one minute and the delay between outputs was averaged. In case of the smaller instances, the faster algorithms were often able to enumerate the whole MEC, for larger instances this is not realistically possible due to their vast size.

For the experiments, we used a single core of the AMD Ryzen Threadripper 3970X 32-core processor on a 256GB RAM machine.[3] We report the times in milliseconds and excluded averages over 32ms from the plots to keep the results visually comparable.

All results in Fig. 7 show the same pattern, namely that MCS-ENUM is superior to the other algorithms and both CHICKERING-ENUM and SHD3-ENUM outperform MEEK-ENUM. The superiority of MCS-ENUM can be easily explained by the fact that its competitors have cost at least in the size of the graph $O(n + m)$ after every (re)-orientation of an edge. While this is the *total cost* between two outputs for MCS-ENUM. The algorithms CHICKERING-ENUM and SHD3-ENUM have a very similar average delay, only the order of the output differs (in the way that SHD3-ENUM guarantees smoothly changing DAGs). MEEK-ENUM has by far the highest cost, due to the large effort arising from the repeated completion under the Meek rules.

Finally, we take a look at the distribution of delays, which we also recorded during the measurements. In Table 1, the percentage of delays d with $d \leq k \cdot \mathrm{mean}$ is given for each algorithm and different k in every considered scenario.

It becomes evident that more than 90 percent of the measured delays are not larger than two times the mean. For the algorithms CHICKERING-ENUM, MCS-ENUM, and SHD3-ENUM, the proportion is even bigger, often between 98 and 99 percent. MEEK-ENUM has the biggest proportion of delays being larger than twice the mean, which can be explained by the fact that some sub-steps need significantly more time to apply Meek's rules than others. Outliers being larger than seven times the mean are nearly as often as outliers being larger than ten times the mean, showing that there are few (in fact, less than 0.5 percent) delays that are way above the mean. We conjecture that these infrequent outliers might be due to garbage collection during the measurements (which becomes a noticeable factor in millisecond measurements).

[3]The experiments can also be run on a desktop computer without any problems.

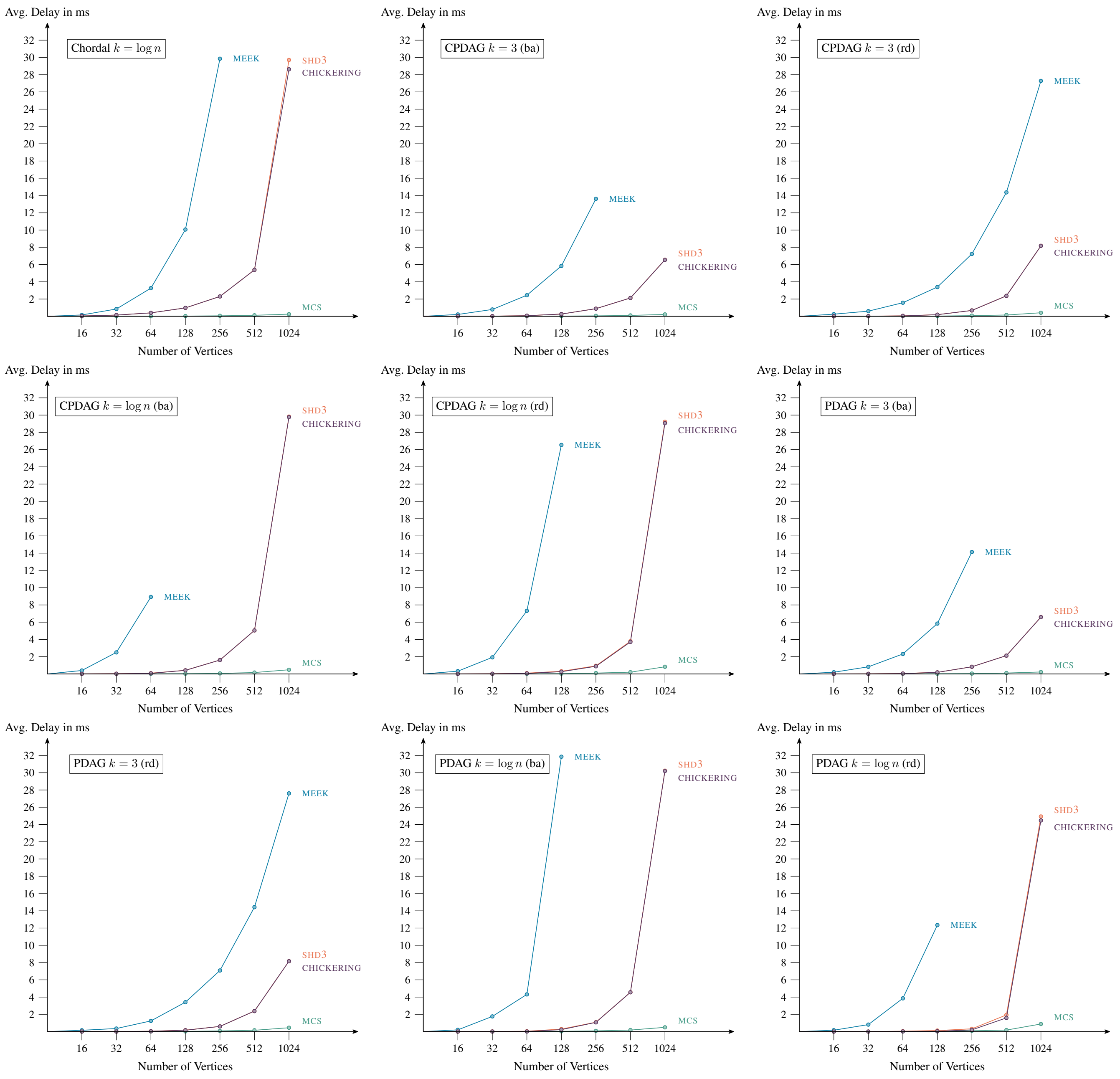

Figure 7: Comparison of the average delay between two output DAGs for the algorithms MEEK-ENUM, CHICKERING-ENUM, MCS-ENUM, and SHD3-ENUM. The graphs are generated as described in the text with density parameter k and the generation methods (ba), for scale-free graphs, and (rd), for uniformly random edge insertions.

Algorithm	Scenario	$k = 1$	$k = 2$	$k = 3$	$k = 5$	$k = 7$	$k = 10$
MEEK-ENUM	UCCG	69.56	91.26	96.69	98.86	99.25	99.68
	CPDAG	63.37	90.52	97.27	99.74	99.95	99.98
	PDAG	63.61	90.79	97.35	99.75	99.95	99.98
	All	69.35	91.24	96.71	98.89	99.27	99.69
CHICKERING-ENUM	UCCG	89.83	99.73	99.86	99.87	99.88	99.89
	CPDAG	86.04	99.77	99.79	99.80	99.81	99.83
	PDAG	81.58	99.73	99.79	99.81	99.83	99.85
	All	89.02	99.73	99.85	99.86	99.87	99.88
MCS-ENUM	UCCG	91.98	99.68	99.77	99.80	99.81	99.81
	CPDAG	76.87	98.93	99.60	99.66	99.66	99.66
	PDAG	74.29	98.42	98.49	99.63	99.63	99.63
	All	89.96	99.56	99.75	99.78	99.79	99.79
SHD3-ENUM	UCCG	77.39	99.37	99.86	99.87	99.88	99.89
	CPDAG	67.81	98.90	99.79	99.80	99.81	99.83
	PDAG	63.35	98.43	99.79	99.81	99.83	99.85
	All	75.73	99.27	99.85	99.86	99.87	99.88

Table 1: The proportion of delays (in percent) that are less or equal to multiples of the mean delay $\bar{d}$, i. e., the percentage of delays d for which $d \leq k \cdot \bar{d}$ holds. We evaluated the delays for each algorithm in a specific scenario and for all scenarios combined.

References

Albert, R.; and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1): 47–97.

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.

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.

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.

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.

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