Extendability of Causal Graphical Models: Algorithms and Computational Complexity

Marcel Wienöbst[1] **Max Bannach**[1] **Maciej Liśkiewicz**[1]

[1]Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

Finding a consistent DAG extension for a given partially directed acyclic graph (PDAG) is a basic building block used in graphical causal analysis. In 1992, Dor and Tarsi proposed an algorithm with time complexity $O(n^4)$, which has been widely used in causal theory and practice so far. It is a long-standing open question whether an extension can be computed faster and, in particular, it was conjectured that a linear-time method may exist. The main contributions of our work are two-fold: Firstly, we propose a new algorithm for the extension problem for PDAGs which runs in time $O(n^3)$; secondly, we show that, under a computational intractability assumption, our cubic algorithm is optimal. Thus, our impossibility result disproves the conjecture that a linear-time method exists. Based on these results, we present a full complexity landscape for finding extensions in various causal graphical models. We extend the techniques to recognition problems and apply them to design an effective algorithm for closing a PDAG under the orientation rules of Meek.

1 INTRODUCTION

Directed acyclic graphs (DAGs) are one of the most fundamental graphical models used in causal analysis. They allow an intuitive and mathematically sound formalism for representing and reasoning about causal knowledge [Spirtes et al., 2000, Pearl, 2009]. However, since in practice the underlying DAGs are unknown or uncertain to the researcher, a crucial task is to learn such structures from data. In particular, an important issue here is to decide whether there exists a causal explanation for the given data at all.

Verma and Pearl [1992] initiated systematic research in this direction by proposing an algorithm for deciding if a set of observed independencies over random variables has an explanation expressed by a causal DAG. The algorithm first extracts as much information as possible from the independence statements and constructs a structure in form of a partially direct acyclic graph (PDAG). Such structures contain directed *and* undirected edges, but do *not* contain any directed cycles. Next, the algorithm extends the PDAG to a consistent, i. e., Markov equivalent DAG, if the PDAG admits such a DAG extension. Finding consistent extensions turned out to be an important building block, which is also required by subsequent learning methods [Meek, 1995, Spirtes et al., 2000, Chickering, 2002]. It is commonly used in software packages for causal analysis [Scutari, 2010, Kalisch et al., 2012, Textor et al., 2016]. In our paper, we investigate this and related problems from an algorithmic and complexity-theoretical perspective.

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

- Firstly, we propose a new algorithm for the extension problem for PDAGs which runs in time $O(n^3)$, or, more precisely, in time $O(\Delta m)$. Moreover, it solves the problem for d-degenerate[3] graphs in time $O(dm)$.

[1]Originally, $O(nm) \in O(n^3)$ was claimed incorrectly. A refined analysis was given by Chickering [2002].

[2]PDX stands for the consistent extension problem for PDAGs.

[3]For a definition, see Sec. 4.

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

- Secondly, using fine-grained complexity analysis, we show that our algorithm is optimal under the *Strong Triangle Conjecture*.

Thus, under the above computational intractability assumption, the conjecture of Dor and Tarsi that there exists a linear-time algorithm to find a consistent DAG extension for PDAGs is not true. On the other hand, from our positive result it follows that, e. g., for forests, constant-treewidth, and planar graphs the problem can be solved in linear time.

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

With these results, we obtain the full and precise complexity-theoretic classification of the extension problem on various graphical causal models, including *Completed Partially Directed Acyclic Graphs* (CPDAGs) [Andersson et al., 1997a, Spirtes et al., 2000, Chickering, 2002] and *Chain Graphs* (CGs) [Lauritzen and Wermuth, 1989, van der Zander and Liśkiewicz, 2016], which is presented in Table 1. Linear-time algorithms are already known for CPDAGs and CGs.

Table 1: Lower and upper bounds on the time complexity of the extension problem for PDAGs, CPDAGs, maximally oriented PDAGs, and Chain Graphs. Previously known bounds were proven by †: Dor and Tarsi [1992] (see also [Chickering, 2002]), *: Hauser and Bühlmann [2012] and ‡: Andersson et al. [1997b]. Our novel lower bound from Corollary 3.4 holds for combinatorial algorithms under the Strong Triangle Conjecture. Bounds in gray are trivial.

Model	*Time Complexity*		*Source*
	Upper Bound	*Lower Bound*	
PDAG	$O(n^4)$		†
	$O(n^3)$		Thm. 4.6
		$\Omega(n^{3-o(1)})$	Cor. 3.4
CPDAG	$O(n+m)$	$\Omega(n+m)$	*
MPDAG	$O(n^4)$	$\Omega(n+m)$	†
	$O(n+m)$		Thm. 5.6
CG	$O(n+m)$	$\Omega(n+m)$	‡

In order to leverage the linear-time algorithms for CPDAGs, MPDAGs and Chain Graphs, it is necessary to check whether a graph belongs to one of these classes. Hence, we expand our analysis to the recognition problem for these graphs. Interestingly, the lower-bound techniques can also be applied in this setting – yielding an $\Omega(n^{3-o(1)})$ bound for MPDAG recognition. We match this bound with an algorithm that recognizes MPDAGs in time $O(n^3)$. Table 2 summarizes the complexities of the recognition problems.

Table 2: A summary for the complexity of the recognition problem for the models as in Table 1.

Model	*Time Complexity*		*Source*
	Upper Bound	*Lower Bound*	
PDAG	$O(n+m)$	$\Omega(n+m)$	
CPDAG	$O(n+m)$	$\Omega(n+m)$	Obs. 6.2
MPDAG	$O(n^3)$		Thm. 6.1
		$\Omega(n^{3-o(1)})$	Cor. 3.4
CG	$O(n+m)$	$\Omega(n+m)$	Obs. 6.2

Finally, we combine our new algorithmic techniques to design an effective method for closing a PDAG under the orientation rules of Meek [1995]. This task of *maximally orienting* a PDAG is an important primitive used in algorithms for learning causal graphs. It also occurs in other settings, such as enumeration problems [Spirtes et al., 2000, Wienöbst and Liśkiewicz, 2020, Guo and Perković, 2020].

The paper is organized as follows. In Sec. 2, we formally introduce the graph classes and problems considered in this work. We derive lower bounds in Sec. 3 and give efficient algorithms for the extension problem on PDAGs and MPDAG in Sec. 4 and Sec. 5, respectively. In Sec. 6, we study the recognition problems. We apply the new techniques to the problem of maximally orienting a PDAG in Sec. 7. Due to space constraints, some proofs (marked with a ▼) are relocated to the supplementary materials.

2 PRELIMINARIES

We investigate *partially directed graphs,* which are triples $G = (V_G, E_G, A_G)$ consisting of a set of *vertices,* a set of *undirected edges*, and a set of *directed edges* (called *arcs*) $A_G \subseteq V_G \times V_G$. If G is clear from the context, we omit the subscripts in V, E, and A. We stipulate that for all pairs $x, y \in V$ at most one of the following is true: $\{x, y\} \in E$, $(x, y) \in A$, or $(y, x) \in A$. That is, an edge is either undirected or directed in one direction. In these cases, we say x and y are adjacent and write $x \sim_G y$. We count the neighbors of $v \in V$ as $\delta(v) = |\{\, w \mid \{v, w\} \in E \,\}|$, $\delta^+(v) = |\{\, w \mid (v, w) \in A \,\}|$, $\delta^-(v) = |\{\, w \mid (w, v) \in A \,\}|$. The *degree* of v is defined as $\Delta(u) = \delta(u) + \delta^+(u) + \delta^-(u)$, and the *maximum degree* of G as $\Delta(G) = \max_{v \in V} \Delta(v)$. We use the abbreviations $n = |V|$, $m = |E| + |A|$, and $\Delta = \Delta(G)$. All graphs in this paper are *connected*, thus, $m \geq n - 1$. We denote the induced subgraph of G on

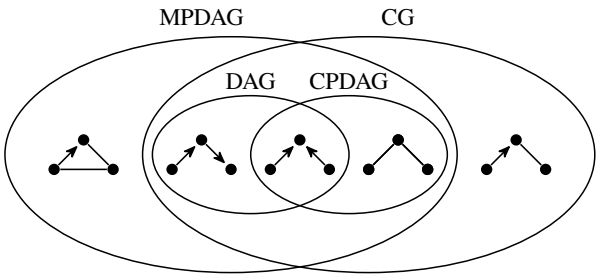

Figure 1: Relation between subclasses of PDAGs: MPDAGs, CGs, CPDAGs, and DAGs.

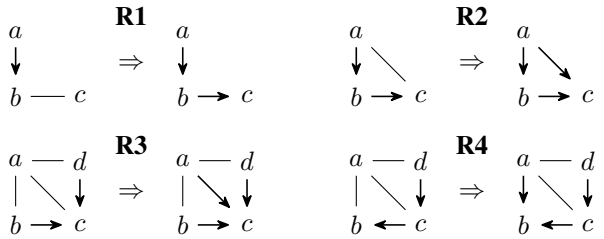

Figure 2: The four Meek rules that are used to characterize MPDAGs [Meek, 1995, Perković, 2020].

a set $C \subseteq V$ by $G[C]$. An *undirected (connected) component* is a maximal undirected connected subgraph (not necessarily induced). A clique is a set of vertices that are pairwise adjacent. A cycle is a sequence of distinct nodes $(c_1, c_2, \ldots, c_k)$, with $k \geq 3$ and edges $c_i - c_{i+1}$ or $c_i \to c_{i+1}$ for $i \in \{1, \ldots, n-1\}$, and $c_k - c_1$ or $c_k \to c_1$. A cycle is *undirected* if all edges are undirected, *directed* if they are all directed, and *semi-directed* if at least one edge is directed. A *v-structure* is an induced subgraph $x \to y \leftarrow z$ with $x \not\sim_G z$. The *skeleton* of $G = (V, E, A)$ is the undirected graph $\big(V, \{\{x, y\} \mid x \sim_G y\}\big)$. An undirected graph is called *chordal* if contains no induced subgraph, which is a cycle of length larger or equal to four.

Directed acyclic graphs (DAGs) have $E = \emptyset$ and contain no directed cycle. Each DAG has a (not necessarily unique) *topological ordering*: a permutation π of the vertices such that u precedes v in π for every edge $u \to v$. DAGs can be partitioned into Markov equivalence classes, which contain DAGs representing the same conditional independence relations [Pearl, 2009]. Verma and Pearl [1990] showed that two DAGs are in the same Markov equivalence class if, and only if, they have the same skeleton and v-structures.

We consider various graphical models: *PDAGs* (partially directed acyclic graphs) are partially directed graphs without a directed cycle; *Chain Graphs* (CGs) are partially directed graphs that do not contain a semi-directed cycle; *MPDAGs* (maximally oriented PDAGs) are PDAGs that are closed under the four *Meek rules* illustrated in Fig. 2. *CPDAGs* (completed PDAGs) represent a Markov equivalence class $\mathcal{C}$. They contain an undirected edge $\{u, v\} \in E$ if there are DAGs $D_1, D_2 \in \mathcal{C}$ with $(u, v) \in A_{D_1}$ and $(v, u) \in A_{D_2}$; and a directed edge $(u, v) \in A$ if there is a D_1 but no D_2. The relations between these classes are illustrated in Fig. 1.

Any partially directed graph $G = (V, E, A)$ represents a possibly empty subset of a Markov equivalence class consisting of DAGs with the same arcs, the same skeleton, and the same v-structures as G. These DAGs are called *consistent (DAG) extensions*[4] and are defined as orientations $\vec{E}$ of E such that $(V, \emptyset, \vec{E} \cup A)$ is *acyclic* and contains the same v-structures as G. Let $CE(G)$ denote the set of all consistent DAG extensions of G. We call G extendable if $CE(G) \neq \emptyset$.

Problem 2.1. EXT

Instance: *A partially directed graph $G = (V, E, A)$.*
Result: *A consistent DAG extension of G if G is extendable; otherwise $\perp$.*

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

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

Observation 2.2 (Dor and Tarsi [1992])**.** *Let V be a vertex set, E be a set of undirected edges, and $A \subseteq V \times V$ be a set of directed arcs. Then:*

(i) $(V, E, \emptyset)$ *is extendable* $\Leftrightarrow$ (V, E) *is chordal*;
(ii) $(V, \emptyset, A)$ *is extendable* $\Leftrightarrow$ (V, A) *is acyclic*.

Another fundamental problem is to recognize classes of partially directed graphs. We define PDAG-REC, CPDAG-REC and CG-REC analogously to:

Problem 2.3. MPDAG-REC

Instance: *A partially directed graph $G = (V, E, A)$.*
Question: *Is G an MPDAG?*

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

Problem 2.4. MAXIMALLY-ORIENT

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

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

[4]For readability, we simply use the term *extensions*.

3 LOWER BOUNDS UNDER THE STRONG TRIANGLE CONJECTURE

In this section, we derive lower bounds for MPDAG-REC, MAXIMALLY-ORIENT and the decision variants of PDAG-EXT and EXT. In particular, the question of whether PDAGs can be extended in linear time has been debated in the past [Dor and Tarsi, 1992], as it is well-known that the related problem of testing chordality can be solved in time $O(n+m)$, i. e., linear in the number of vertices and edges [Rose et al., 1976]. We show that in order to extend PDAGs for arbitrary E and A in linear time, however, a great algorithmic breakthrough would be necessary.

We prove these results through reductions from the problem TRIANGLE (does an undirected graph contain three pairwise connected vertices?). It is conjectured that, for any $\varepsilon > 0$, there is no algorithm that solves TRIANGLE in time $O(n^{\omega-\varepsilon})$ (were $\omega < 2.373$ is the matrix multiplication exponent) and no combinatorial algorithm running in time $O(n^{3-\varepsilon})$. Here, combinatorial means "not using algebraic techniques" and the distinction is made as algebraic algorithms are often not well-suited for practical applications[5].

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

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

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

The reductions are based on the following idea: We partition the graph into three parts such that we have control over the structure of possible triangles; then we direct only a few edges such that detecting induced subgraphs

[5] In the algorithmic literature, the term combinatorial algorithm is mainly used for distinguishing those approaches which are different from the algebraic approach originated with the work of Strassen. In particular, one can simply think of a combinatorial algorithm as such that does not call an oracle for ring matrix multiplication. For more details, see [Williams and Williams, 2018].

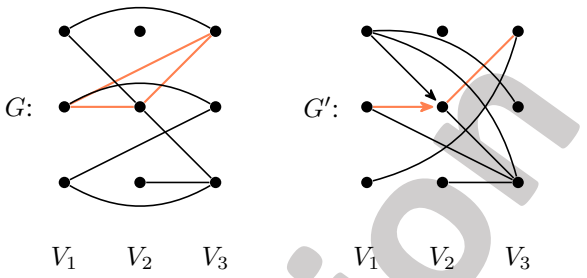

Figure 3: The basic step of the reductions: The edges between V_1 and V_2 are oriented towards V_2 and the edges between V_1 and V_3 are complemented. A triangle in G implies the subgraph $a \to b - c$ in G' and vice versa.

$a \to b - c$ corresponds to finding triangles. Our first task is, thus, to prove that TRIANGLE is at least as hard as 3PART-TRIANGLE (does a 3-partite undirected graph contain a triangle?). An undirected graph is k-partite if there is a partition $V = V_1 \dot\cup V_2 \dot\cup \cdots \dot\cup V_k$ such that there is no edge $\{u, v\} \in E$ with $u, v \in V_i$ for some i.

Lemma 3.1 (▼). *There is an $O(n^2)$ time reduction from* TRIANGLE *to* 3PART-TRIANGLE *that increases the number of vertices only by a constant factor.*

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

Theorem 3.2. *There is an $O(n^2)$ time reduction from* TRIANGLE *to the* MPDAG-REC *problem that increases the number of vertices only by a constant factor.*

Proof. First apply Lemma 3.1 to reduce TRIANGLE to 3PART-TRIANGLE and let $G = (V, E, \emptyset)$ be the resulting instance with partitions $V_1 \cup V_2 \cup V_3 = V$. Next, transform this instance as follows.

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

$$E' = \{\, \{u, v\} \mid \{u, v\} \in E \text{ with } u \in V_2 \text{ and } v \in V_3 \,\} \cup \{\, \{u, v\} \mid \{u, v\} \notin E \text{ with } u \in V_1 \text{ and } v \in V_3 \,\};$$
$$A' = \{\, (u, v) \mid \{u, v\} \in E \text{ with } u \in V_1 \text{ and } v \in V_2 \,\}.$$

Next, construct a partially directed graph $G_{\text{rec}} = (V, E' \cup \{\, \{u, v\} \mid u \neq v \in V_1 \,\}, A')$, i. e., modify the construction shown in Fig. 3 by additionally pairwise connecting V_1 with undirected edges. We show that G contains a triangle if, and only if, G_{rec} is not an MPDAG:

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

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

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

From Theorem 3.2 we can deduce immediately that under STC any combinatorial algorithm solving MPDAG-REC problem requires $\Omega(n^{3-o(1)})$ time. The same lower bound holds for MAXIMALLY-ORIENT. Otherwise, one could solve MPDAG-REC in sub-cubic time by applying the algorithm for MAXIMALLY-ORIENT to the given graph G and testing if the resulting graph coincides with G.

Theorem 3.3 (▼). *There is an $O(n^2)$ time reduction from* TRIANGLE *to the decision variant of* PDAG-EXT *that increases the number of vertices only by a constant factor.*

Sketch of Proof. Modify the graph G' constructed in the proof of Theorem 3.2 and illustrated in Figure 3 to G_{ext} as folllows. Complete the vertices in V_3 to an undirected clique and add a new vertex z with directed edges $z \to v \in V_3$. A triangle in G becomes $a \to b - c \leftarrow z$ in G_{ext}. As the edge $b - c$ cannot be oriented without creating a new v-structure, G_{ext} is not extendable. If graph G does not contain a triangle, one can construct a consistent extension by orienting the undirected edges as follows: $V_3 \ni u \to v \in V_1$; $V_3 \ni u \to v \in V_2$; and the edges in V_3 according to some arbitrary linear ordering. □

Corollary 3.4. *For any $\varepsilon > 0$, every algorithm that solves* MPDAG-REC, MAXIMALLY-ORIENT, PDAG-EXT, *or* EXT *in time $O(n^{\omega-\varepsilon})$ violates STC. Any combinatorial algorithm that solves one of these problems in time $O(n^{3-\varepsilon})$ also violates the conjecture.*

4 EXTENDING PARTIALLY DIRECTED GRAPHS

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

A *graph property* Π is a family of graphs that is closed under isomorphism, e. g., being chordal is a graph property. We say Π is *hereditary* if it is also closed under taking induced subgraphs. For instance, all three properties (being extendable, chordal, or acyclic) are hereditary.

Definition 4.1. *Let p be a property of a vertex. A p-*elimination-order *of a (directed) graph $G = (V, E)$ is an order π such that every vertex v has property p in the induced subgraph $G[V \setminus \{ w \mid \pi(w) < \pi(v) \}]$.*

Elimination orders characterize hereditary graph classes. For instance, a vertex v in an undirected graph is called *simplicial* if its neighbors $N(v)$ form a clique; a vertex w in a directed graph is a *sink* if it has no outgoing arc. It is well-known that a (directed) graph is chordal iff it has a simplicial-elimination-order; and is acyclic iff it has a sink-elimination-order [Fulkerson and Gross, 1965]. Dor and Tarsi [1992] observed that these properties can be combined to obtain a characterization of extendable graphs:

Definition 4.2. *A* potential-sink *in a partially directed graph $G = (V, E, A)$ is a vertex v s. t. $X = \{ w \mid \{v, w\} \in E \}$ is a clique, $\{ w \mid (v, w) \in A \}$ is empty, and $x \sim_G y$ for all $x \in X$ and $y \in Y = \{ w \mid (w, v) \in A \}$.*

Fact 4.3 (Dor and Tarsi, 1992)**.** *Every partially directed graph that is extendable contains a potential-sink.*

Corollary 4.4. *A partially directed graph is extendable iff it has a potential-sink-elimination-order.*

Proof. The first direction follows from Fact 4.3 as extendable graphs are hereditary. On the other hand, successively removing an potential-sink and orienting all its incident edges towards it yields an extension of the graph. □

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

Definition 4.5. *A graph $G = (V, E)$ is* d-degenerate *if there is a vertex-ordering $\prec$ (called* degeneracy ordering*) such that we have $|\{ w \mid w \prec v \text{ and } w \sim_G v \}| \leq d$ for every $v \in V$. The smallest value d for which G is d-degenerate is the* degeneracy *of G.*

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

Theorem 4.6. *There is an algorithm that decides whether a PDAG G is extendable in time $O(dm)$. If G is extendable, a consistent extension can be computed within the same time bound. Here, d is the degeneracy of the skeleton of G.*

The proof of the theorem is based on the following slightly weaker version, which only achieves $O(\Delta m)$.

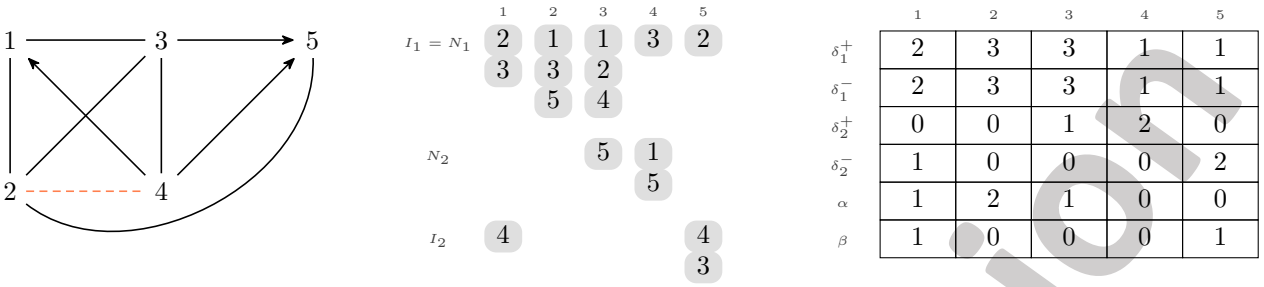

Figure 4: The data structure to represent PDAGs $G = (V, E, A)$ is based on a hybrid graph representation by Abu-Khzam et al. [2010]. We maintain two directed graphs $G_1 = (V, E)$ and $G_2 = (V, A)$, where G_1 is symmetric. Each G_i is represented by an $n \times n$ adjacency matrix M_i (not displayed here), and two adjacency lists I_i and N_i, storing the incoming and outgoing arcs of each vertex. We also store for each vertex its in- and out-degree $\delta_i(v)^-$ and $\delta_i(v)^+$. The clue is that in $M_1[x, y] = \ell$ we store the position ℓ of an edge $\{x, y\}$ in the adjacency list $I_1[x]$, respectively, $M_2[x, y] = (\ell_1, \ell_2)$ stores the position ℓ_1 of an edge (x, y) in the adjacency list $N_2[x]$ and ℓ_2 the position of (x, y) in $I_2[y]$. Hence, we have $O(1)$ adjacency checks, edge insertion, and edge deletion. Note that, in practice, one may also use hash tables to store the adjacencies of the vertices in G_1 and G_2, with *expected* time $O(1)$ for the stated operations. To manage potential-sinks, we use arrays α, β that track the edges in the neighborhood of a vertex: $\alpha[v] = |\{\, \{x, y\} \mid \{v, x\} \in E \wedge \{v, y\} \in E \wedge x \sim_G y \,\}|$ and $\beta[v] = |\{\, \{x, y\} \mid \{x, v\} \in E \wedge (y, v) \in A \wedge x \sim_G y \,\}|$. We initialize these values to zero and update them in time $O\big(\delta_1^+[u] + \delta_2^+[u] + \delta_2^-[u]\big)$ for edge insertions and deletions. A vertex $s \in V$ is a potential-sink iff (i) $\alpha[s] = \binom{\delta_1^+[v]}{2}$; (ii) $\beta[s] = \delta_1^+[s] \cdot \delta_2^-[s]$; and (iii) $\delta_2^+[s] = 0$. Therefore, the data structure can support the claimed operations. The figure shows this data structure for the partially directed graph at the top left (without the dotted edge). The adjacency lists are arrays to be read from top to bottom. Neither 1 nor 5 are potential-sinks (in both cases the edge $2 \sim 4$ is missing). If we insert the dotted edge we get $\beta[1] = \beta[5] = 2$, $\delta_2^+[1] = \delta_2^+[5] = 0$, $\alpha[1] = \binom{2}{2} = 1$ and $\alpha[5] = \binom{1}{2} = 0$ – hence, both become potential-sinks by the sketched criterion.

Lemma 4.7 (▼). *There is an algorithm that decides whether a PDAG G is extendable in time $O(\Delta m)$.*

Sketch of Proof. The backbone of the algorithm is a data structure that maintains PDAGs and potential-sinks therein. It supports insertion and deletion of edges $\{u, v\}$ or arcs (u, v) in $O(\delta(u))$; can test whether a vertex is a potential-sink in $O(1)$; and provides the operation pop-ps(s) that orients all incident edges towards a potential-sink s, removes s, and returns all neighbors of s that did become a potential-sink by this operation – all in time $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$. A full implementation is provided in the supplementary material; Fig. 4 contains a brief illustration of it.

Given this tool, the algorithm becomes rather simple: Initialize the data structure by inserting all edges (in time $O(\Delta m)$); then initialize a stack of all potential-sinks by testing for every vertex whether it is a potential-sink (overall $O(n)$); finally, as long as there is potential-sink s on the stack, apply pop-ps(s), remove s from the stack, and push the newly generated potential-sinks to the stack. The overall run time is obtained as follows: pop-ps(s) runs in time $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$ and removes $\delta^-(s) + \delta(s)$ edges from the graph. By rewriting $\delta(s)^2 + \delta(s) \cdot \delta^-(s)$ as $\delta(s) \cdot \big(\delta(s) + \delta^-(s)\big)$ we see that we pay $O(\delta(s)) \leq O(\Delta)$ per edge. Since the algorithm terminates after the removal of all edges, an overall run time of $O(\Delta m)$ follows. □

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

Proof of Theorem 4.6. Before we initialize the data structure, we compute a degeneracy ordering of the skeleton of the graph. This is possible in time $O(n + m)$ with the algorithm of Matula and Beck [1983]. Let the ordering be $v_1, v_2, \ldots, v_n$, i. e., v_i has at most d neighbors in $v_1, \ldots, v_{i-1}$. We iterate through the vertices and insert their incident edges to preceding neighbors. That is, if we handle a vertex v_i, we insert all edges $\{v_i, v_x\}$ and arcs (v_i, v_y), (v_z, v_i) with $x, y, z < i$. In this way, v_i has degree at most d if we insert an edge for it and, thus, inserting the edge requires time $O(d)$, yielding an initial time of $O(dm)$.

We proceed as in the proof of Lemma 4.7 and maintain a stack of potential-sinks. While the stack is not empty, we use pop-ps(s) to orient edges towards s and remove s from the graph. Recall that this costs $O(\delta(s)^2 + \delta(s) \cdot \delta^-(s))$.

Claim 4.8 (▼). *A potential-sink s in a d-degenerate graph has at most $d + 1$ undirected neighbors.*

Table 3: Running time of the algorithm from Theorem 4.6 on graphs whose skeleton is in the mentioned graph class.

Graph Class	*Time*	*Note*
d-degenerate	$O(dm)$	Theorem 4.6
general graphs	$O(nm)$	$d \leq n$
forests	$O(m)$	constant degeneracy
series-parallel	$O(m)$	
planar	$O(m)$	
treewidth-t	$O(tm)$	$d \leq t$

Removing a potential-sink s, thus, produces costs of at most $O(d^2 + d \cdot \delta^-(s))$. Since this removes $\delta(s) + \delta^-(s)$ edges from the graph, we pay $O(dm)$ to remove *all* potential-sinks.

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

Let us close this section by pointing out the advantage of an $O(dm)$ algorithm compared to an $O(\Delta m)$ algorithm. Many natural graph classes have bounded degeneracy, but unbounded maximum degree – for instance, planar graphs and graphs of bounded treewidth. Our algorithm runs in *linear time* on such graphs. Table 3 summarizes the findings of this section on some well-known graph classes.

5 A GRAPHICAL CHARACTERIZATION OF EXTENDABLE MPDAGS

As discussed in the previous section, sink-based reduction algorithms can extend graphs in time $O(dm)$. The lower bounds of Sec. 3 suggests that no significantly faster algorithm is possible. However, the lower bounds do not rule out faster algorithms for more structured graphs. It is well-known that extensions of CPDAGs can be constructed in linear time by algorithms such as the Lexicographic BFS or Maximum Cardinality Search [Rose et al., 1976, Tarjan and Yannakakis, 1984]. These algorithms were originally developed for chordality testing. The connection to CPDAGs is due to the fact that the undirected components of CPDAGs are *chordal* [Andersson et al., 1997a]. Orienting these undirected parts is precisely the task of extending a CPDAG.

A related graph class is the one of MPDAGs, which encode additional background knowledge compared to CPDAGs and are, in contrast to PDAGs, completed by the four Meek rules R1-R4. Unlike CPDAGs, MPDAGs are not extendable by definition – see Fig. 5. For practical use, however, only *extendable* MPDAGs are meaningful. Hence, it is our goal to provide a graphical characterization for such graphs. In addition, this characterization allows us to compute an extension in linear-time. Consider the graph **(2)** from Fig. 5 and observe that in order to extend an MPDAG, it is not sufficient to orient the undirected components (they are not necessarily chordal). In particular, when we study MPDAGs, we have to consider the directed edges, too. Therefore, we make use of a graphical object introduced by Perković [2020]:

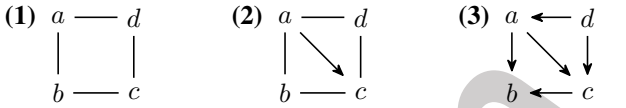

Figure 5: Graph **(1)** is not extendable: Orienting the cycle of length 4 will either create a v-structure or a directed cycle. (Intuitively, this explains why an undirected graph is extendable iff it is chordal.) However, the graph is an MPDAG by definition. Graph **(2)** on the other hand is extendable, a consistent DAG extension is shown in **(3)**.

Definition 5.1. *Let G be an MPDAG and $C = (V_C, E_C, \emptyset)$ an undirected component of G. We call $B = G[V_C]$ a* bucket.

A bucket is the induced subgraph on the vertices of an undirected component and, hence, may contain directed edges such as $a \to c$ in **(2)** of Fig. 5. Using the information available through the directed edges in each bucket, we want to find a criterion for MPDAG extendability. We start by observing the following property:

Lemma 5.2 (▼). *Let G be an MPDAG. A bucket B of G does not contain a v-structure.*

Orienting buckets acyclically without *any* v-structure is an important step towards computing an extension. For such orientations, we use the shorthand AMO (acyclic moral orientation).[6] Indeed, any AMO of the buckets will suffice to find a consistent extension of an MPDAG.

Lemma 5.3 (▼). *Let G be an MPDAG. Computing an AMO for each bucket yields a consistent extension of G.*

The question remains whether every bucket has an AMO. The lemma below provides a graphical criterion to decide whether this is the case. Moreover, we present a linear-time algorithm that computes such an AMO of a bucket.

We note that preliminary results in this direction were obtained by Meek [1995] (Lemma 8). However, these are non-constructive and *assume* extendability.

Lemma 5.4 (▼). *Let G be an MPDAG. A bucket B of G has an AMO if, and only if, its skeleton is chordal. Such an AMO can be computed in linear time.*

[6] We use the term to emphasize the absence of a v-structure; an AMO of a bucket can also be seen as a consistent extension.

Sketch of Proof. We adapt the Lexicographic BFS (LBFS) algorithm [Rose et al., 1976] to find an AMO of B consistent with the directed edges that are already present. It is well-known that the orientation induced by an LBFS traversal order ($a \to b$ if a is visited before b) is an AMO if, and only if, the skeleton is chordal [Hauser and Bühlmann, 2012].

At each step of the LBFS, the algorithm has a candidate set of vertices to explore next. In order to reproduce the directed edges in B, we demand that a vertex is chosen that has no incoming edges from unvisited vertices in B. This way one obtains an AMO, which reproduces the already directed edges. Through the properties of MPDAGs and the LBFS, it can be shown that such a vertex always exists. □

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

Theorem 5.5. *An MPDAG is extendable if, and only if, the skeleton of every bucket is chordal.*

Proof. Combine Lemma 5.3 and Lemma 5.4. □

Theorem 5.6. MPDAG-EXT *can be solved in linear time* $O(n + m)$.

Proof. Perform the modified LBFS from the proof of Lemma 5.4 on each bucket. Afterward, test in linear time whether the orientation is an AMO [Rose et al., 1976]. □

6 RECOGNITION OF CAUSAL GRAPH CLASSES

As shown in the previous section, it is possible to find a consistent extension of an MPDAG in linear-time. The remaining question is how to check whether a partially directed graph is an MPDAG. Checking acyclicity can be done in linear time, but testing the closedness under the Meek rules R1-R4 is harder as the lower bound in Sec. 3 suggests.

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

Theorem 6.1. MPDAG-REC *can be solved in time* $O(\Delta m)$.

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

1. If G contains a directed cycle, return "No".
2. R1: If $\exists\, b - c$ and vertex a s.t. $a \to b - c$, return "No".
3. R2: If $\exists\, a - c$ and vertex b s.t. $a \to b \to c$, return "No".
4. R3: If $\exists\, d \to c$ and vertex a s.t. $a - d \to c$ as well as vertex b s.t. $b \to c \leftarrow d$, return "No".
5. R4: If $\exists\, c \to b$ and vertex d s.t. $d \to c \to b$ as well as vertex a s.t. $a - b \leftarrow c$, return "No".
6. Return "Yes".

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

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

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

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

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

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

Observation 6.2 (▼). *The problems* PDAG-REC, CPDAG-REC, *and* CG-REC *can be solved in time* $O(n + m)$.

7 APPLICATION TO MAXIMAL ORIENTATIONS OF PDAGS

In this section, we study the problem MAXIMALLY-ORIENT, i. e., the task of computing the closure of a PDAG under the orientation rules R1-R4. As we noted in Sec. 5, for applications in causality, only *extendable* models are meaningful. In particular, a non-extendable PDAG has no causal explanation and closing such a graph under the orientation rules R1-R4 is purposeless. This justifies that our algorithm for MAXIMALLY-ORIENT assumes that the given PDAG is extendable. We will see that the ideas of the previous sections

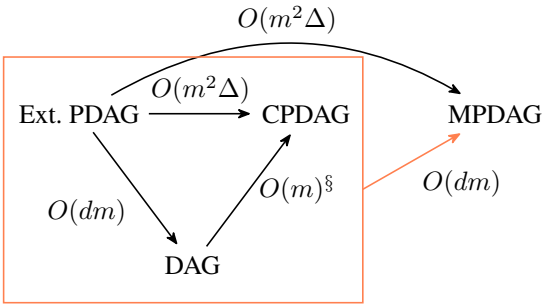

Figure 6: Using the techniques from Sec. 6, it follows that (extendable) PDAGs can be maximally oriented into CPDAGs/MPDAGs in $O(m^2\Delta)$ by direct application of the Meek rules. Using the new PDAG extension algorithm from Sec. 4, PDAG-to-CPDAG can be performed in time $O(dm)$ as DAG-to-CPDAG is possible in $O(m)$ (§: [Chickering, 1995]). In Sec. 7, we give an $O(dm)$ algorithm for PDAG-to-MPDAG, which uses a consistent DAG and the corresponding CPDAG as auxiliary graphs.

can be combined to obtain more efficient algorithms for this problem. An overview of the results is given in Fig. 6.

In general, MAXIMALLY-ORIENT produces an MPDAG when the input is an extendable PDAG. If the directed edges follow from v-structure information, the resulting graph is even a CPDAG. In this setting, Chickering [2002] noticed that, in order to maximally orient a PDAG into a CPDAG, it is not necessary to apply the Meek rules directly. A more efficient computation is possible by first extending the PDAG into a DAG and afterward transforming the DAG into the corresponding CPDAG. The latter task can be performed in linear time [Chickering, 1995]. In combination with the PDAG extension algorithm presented in this work, we can conclude that PDAG-to-CPDAG can be performed in time $O(dm)$ via the route PDAG-DAG-CPDAG. Notably, this is significantly faster than directly applying the Meek rules, which takes time $O(m^2 \cdot \Delta)$ (with the improvements for detecting the Meek rules that we discussed in the previous section).

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

Theorem 7.1 (▼). *For extendable PDAGs, problem* MAXIMALLY-ORIENT *can be solved in time* $O(dm)$.

Sketch of Proof. For an extendable PDAG G, compute a consistent extension D and its CPDAG C. It is sufficient to apply the Meek rules to the vertices in undirected components of C. Consider such an undirected component U and orient edges, which correspond to arcs in G. Afterward, detect and apply the Meek rules by traversing V_U according to a topological ordering of $D[V_U]$. Each vertex in $D[V_U]$ has at most d parents, as these have to form a clique (there can be no v-structures in U) and d-degenerate graphs contain cliques of size at most $d + 1$. Through this observation, it is possible to bound the run time by $O(dm)$. □

8 CONCLUSIONS

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

Acknowledgements

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

References

Faisal N. Abu-Khzam, Michael A. Langston, Amer E. Mouawad, and Clinton P. Nolan. A hybrid graph representation for recursive backtracking algorithms. In *FAW*, volume 6213 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2010.

Steen A. Andersson, David Madigan, and Michael D Perlman. A characterization of Markov equivalence classes

for acyclic digraphs. *The Annals of Statistics*, 25(2): 505–541, 1997a.

Steen A Andersson, David Madigan, and Michael D Perlman. On the Markov equivalence of chain graphs, undirected graphs, and acyclic digraphs. *Scandinavian Journal of Statistics*, 24(1):81–102, 1997b.

David Maxwell Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proc. of the Conference on Uncertainty in Artificial Intelligence, UAI 1995*, pages 87–98, 1995.

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

Dorit Dor and Michael Tarsi. A simple algorithm to construct a consistent extension of a partially oriented graph. *Technicial Report R-185, Cognitive Systems Laboratory, UCLA*, 1992.

D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3): 835–855, 1965.

F Richard Guo and Emilija Perković. Minimal enumeration of all possible total effects in a Markov equivalence class. *arXiv preprint arXiv:2010.08611*, 2020.

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.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47(11):1–26, 2012.

Steffen Lauritzen and Nanny Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17:31–57, 1989.

David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

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

Judea Pearl. *Causality*. Cambridge University Press, 2009.

Emilija Perković. Identifying causal effects in maximally oriented partially directed acyclic graphs. In *Proc. of the Conference on Uncertainty in Artificial Intelligence, UAI 2020*, volume 124 of *Proceedings of Machine Learning Research*, pages 530–539. AUAI Press, 2020.

Emilija Perković, Markus Kalisch, and Marloes H Maathuis. Interpreting and using cpdags with background knowledge. In *Proc. of the Conference on Uncertainty in Artificial Intelligence, UAI 2017*. AUAI Press, 2017.

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

M Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3), 2010.

Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

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

Johannes Textor, Benito van der Zander, Mark S Gilthorpe, Maciej Liśkiewicz, and George TH Ellison. Robust causal inference using directed acyclic graphs: the R package dagitty. *International journal of epidemiology*, 45(6): 1887–1894, 2016.

Benito van der Zander and Maciej Liśkiewicz. Separators and adjustment sets in Markov equivalent dags. In *Proc. of the Conference on Artificial Intelligence, AAAI 2016*, pages 3315–3321, 2016.

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

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

Marcel Wienöbst and Maciej Liśkiewicz. Recovering causal structures from low-order conditional independencies. In *Proc. of the AAAI Conference on Artificial Intelligence, AAAI 2020*, volume 34, pages 10302–10309. AAAI Press, 2020.

Virginia Vassilevska Williams and R Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.