UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Constraint-based causal structure learning exploiting low-order conditional independences

Einschränkungsbasiertes Lernen von kausalen Strukturen durch Nutzung von bedingten Unabhängigkeiten geringer Ordnung

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Marcel Wienöbst

ausgegeben und betreut von
Prof. Dr. Maciej Liśkiewicz

Lübeck, den 8. Juni 2019

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

Lübeck, den 8. Juni 2019

Danksagungen

Zuerst möchte ich Prof. Maciej Liśkiewicz für seinen Rat beim Schreiben dieser Arbeit danken. Ein besonderer Dank gilt auch Katharina Dannenberg für erkenntnisreiche Gespräche über das Lernen von kausalen Strukturen.

Zudem bedanke ich mich bei meinem Kommilitonen Thore Tiemann und meinem Bruder für das Korrekturlesen.

Abschließend möchte ich meinen Eltern für die durchgängige Unterstützung während meines Studiums danken.

Abstract

This Master's thesis deals with the topic of learning causal structure from conditional independences. Of particular interest are sets of conditional independences with a low order. For the special case of marginal independences we give important proofs which were missing from the literature. For the until now unsolved, general case of conditional independences up to a fixed order k we propose a new algorithm which finds the faithful DAGs and prove its correctness. The main result is the application of the new concepts to the problem of causal structure learning from complete conditional independence information. This enables us to propose a constraint-based algorithm which learns causal structures more efficiently than the popular PC algorithm.

Zusammenfassung

Diese Masterarbeit behandelt die Fragestellung, wie kausale Strukturen durch bedingte Unabhängigkeiten gelernt werden können. Von besonderem Interesse sind Mengen, bei denen die Ordung der bedingten Unabhängigkeiten gering ist. Für den Randfall von unbedingten Unabhängigkeiten ergänzen wir den derzeitigen Stand der Forschung um wichtige Beweise. Für den bislang ungelösten, allgemeinen Fall von bedingten Unabhängigkeiten bis zu einer bestimmten Ordnung k geben wir einen Algorithmus, der die konsistenten DAGs findet, und beweisen dessen Korrektheit. Das Hauptresultat ist die Anwendung der neuen Konzepte auf das Lernen von kausalen Strukturen für den Fall, dass alle bedingten Unabhängigkeiten zur Verfügung stehen. Daraus leiten wir einen Algorithmus her, der kausale Strukturen effizienter als der weit verbreitete PC Algorithmus lernt.

Contents

1 Introduction

In empirical sciences researchers are interested in discovering causal relationships. To analyze such relationships the interdisciplinary research field of causal inference has been developed [13]. In particular, directed acyclic graphs (in short DAGs) are used to model causal relationships. These DAGs are also called causal structures. The nodes of the causal structure represent random variables and the directed edges causal dependences. These causal structures enable, among others, the estimation of causal effects and the analysis of counterfactuals. Causal inference is especially useful when experimental data cannot be obtained, for example, for ethical reasons.

A common scenario is that such causal structures are specified by an expert. However, for more complex systems this is infeasible and, in particular, for Artificial Intelligence applications it is vital that this process is automated. A central question is therefore how causal structures can be learned from observed data. This problem is known to be NP-hard [4]. Still, there are algorithms available for practical use. One of the main classes of causal structure learning algorithms are the constraint-based approaches. These algorithms learn the causal structure by searching for conditional independences (CIs). Such a CI implies that the nodes corresponding to the conditionally independent random variables are nonadjacent in every faithful DAG. A DAG is faithful if it entails exactly the observed conditional independences. This way it is possible to discover the causal structure. The PC algorithm [8, 17] is the most popular of the constraint-based algorithms. It runs in time polynomial in the maximal node degree of the underlying DAG and is able to find the exact solution under certain assumptions. One of these assumptions is that the PC algorithm has access to the true conditional independence information. However, in practice these conditional independences have to be tested statistically.

Testing for conditional independences is inherently more difficult than testing for marginal independences [3] due to the fact that the variables could be dependent for only a few values of the conditioning variable. This problem increases in significance for larger conditioning sets and, therefore, tests in this setting require a high number of observations which are often not available in practice. Inaccurate estimations of CIs are a consequence that may lead to a high number of false positives or false negatives for the classification of direct causal relations. A possible way to circumvent this problem is to primarily consider low-order CIs and extract as much information as possible from these statements. Low-order CIs are those where the cardinality of the conditioning set is small, for example one or two.

The current state of research on the topic of causal structure learning with an emphasis on low-order CIs leaves many problems open. Most of the work focused on finding faithful DAGs for the important special case of marginal independences. Pearl and Wermuth [14] proposed an algorithm which finds a certain class of faithful DAGs. This algorithm was extended by Textor et al. [18] in order to find all faithful DAGs. But in both works the correctness of the algorithm was not proven. Furthermore, it is unclear how the approaches

proposed for the case of marginal independences can be generalized to sets of low-order CIs. Campos and Huete [7] consider CIs with order one, but they presuppose knowledge of the topological sorting of the underlying DAG. Wille and Bühlmann [21] also deal with CIs of order one and propose an algorithm which finds faithful DAGs. However, this algorithm is only correct for certain graph classes. Finally, the question remains open how these concepts might be applied to causal structure learning in general.

1.1 Main results

This thesis deals with exactly these unsolved problems. We begin by investigating how causal structures can be recovered when only low-order CIs are available. We approach this problem from a causal structure learning perspective. As mentioned above, the constraint-based algorithms rule out edges when finding corresponding conditional independences. This is possible because these edges cannot be in a faithful DAG. In this spirit, we introduce the notion of the *representation* of the set of faithful DAGs. The representation contains an edge if and only if this edge is present in at least one faithful DAG. Vice versa, edges which are in no faithful DAG are not in the representation. Thus, our goal is to classify and remove these edges. We will later see that obtaining the representation also enables us to find the faithful DAGs.

We reexamine the problem of recovering causal structures from marginal independences and prove the correctness of the previously proposed algorithms (Theorem 2 in [14] and Theorem 4.4 in [18]). We do this by presenting an equivalent algorithm which is simpler and easier to generalize than the construction by Pearl and Wermuth [14]. Moreover, we show that the resulting graph coincides with the representation.

Based on this we tackle the generalized and until now unsolved problem of recovering causal structures from sets of CIs whose order is bounded by a fixed value k. We introduce an algorithm which solves this problem by finding the representation. This algorithm generalizes our results on marginal independences. Moreover, we gain insights into the set of faithful DAGs by showing that the representation is always a CPDAG — a *Completed partially directed acyclic graph* is the standard way of representing a Markov equivalence class. The notion was first developed by Meek [12] and Andersson et al. [2]. This enables us to view the concept of a representation as a generalization of the concept of a CPDAG.

Afterwards, we apply the new ideas developed for the learning from low-order CIs to causal structure learning in general by proposing an improved version of the popular PC algorithm. This is the main result of this thesis. The essential idea behind this improvement is the following. The PC algorithm (and every other constraint-based causal structure learning algorithm — see Chapter 2.5 in the book by Pearl [13]) is made up of three steps:

1. Learning the skeleton.
2. Orientation of the v-structures (obtaining a pattern).
3. Orientation of the remaining edges.

The PC algorithm begins by learning the skeleton which is an undirected graph storing the adjacency information of the underlying causal structure. The edges are oriented only afterwards. The orientation phase itself consists of two steps. First, the v-structures are detected. The obtained intermediate result is also called a pattern. A v-structure is a substructure $i \to k \leftarrow j$ where there are two converging arrows whose tails are not directly connected. Verma and Pearl [19] showed that two DAGs are observationally equivalent if and only if they have the same skeleton and the same set of v-structures. Observational equivalence means that every probability distribution which is compatible with one of the DAGs is also compatible with the other [13]. Therefore, obtaining the pattern which has the same skeleton and the same v-structures as every faithful DAG is a vital step in causal structure learning. Finally, as many undirected edges as possible are oriented in order to extend the graph maximally. The new approach we propose consists only of two steps:

1. Learning the (extended) pattern.
2. Orientation of the remaining edges.

We call this approach the OPPC (One phase pattern construction) algorithm. The central modification is that we merge the first two steps of the PC algorithm — the learning of the skeleton and the detection of the v-structures. This is nontrivial and only possible by using the concepts we develop for the learning of causal structures from low-order CIs. In essence, this is because when recovering faithful DAGs from low-order CIs, it is vital to direct edges before having complete conditional independence information. This is similar to the task of directing edges before knowing the true skeleton.

Most importantly, we show experimentally that by merging the two steps we are able to improve the efficiency of learning the causal structure significantly. For example for regular graphs with 76 nodes and node degree 5, the PC algorithm needs more than 8 times as many CI tests as the OPPC algorithm. This improvement is due to extensive use of low-order CIs as the following numbers confirm. After considering the marginal independences the OPPC algorithm already removed $1,672$ edges compared to only 992 in the PC algorithm. These early removals of edges lead to a substantially smaller search space.

1.2 Structure of the thesis

The following is an overview of the structure of this thesis: In Section 2 we propose the used notation and formally introduce the most essential definitions. Section 3 reviews the most important previous work on the topics dealt with in this thesis. In Section 4 we introduce the new concept of incompatible nodes which plays a key role in recovering causal structures from a set of low-order CIs. Section 5 deals with the special case of recovering causal structures from a set of marginal independences. We give important proofs which have been missing from the literature. The approach for marginal independences is generalized

in Section 6 where we solve the problem of recovering causal structures from a set of CIs up to order k. The concepts are then applied to improve causal structure learning in general (Section 7). In particular, we propose a new constrained-based algorithm which we term OPPC (One phase pattern construction). Afterwards, in Section 8 we experimentally analyze the performance of the OPPC algorithm and compare it to the PC algorithm. Finally, in Section 9 we discuss new questions raised by the results of this thesis and possible future work on the topic of causal structure learning from low-order CIs.

2 Preliminaries

In this section we give the notation used throughout this thesis and introduce some basic definitions. For all remaining definitions we refer the reader to the fundamental book by Pearl [13].

2.1 Graphical Background

We begin by introducing the necessary graphical definitions. A graph $G = (V, E)$ consists of a vertex set V and a set of edges E. We use the variable n to describe the number of vertices: $|V| = n$. We are mainly concerned with directed and partially directed graphs. The latter type describes a graph where edges can be directed as well as undirected. Formally this can be modeled in the following manner:

$$E = \{(i, j) \mid \text{if there is a directed edge from } i \text{ to } j \text{ or an undirected edge between } i \text{ and } j\}$$

This means that in case we have $(i, j) \in E$ and $(j, i) \in E$, there is an undirected edge between i and j. Two nodes i and j are called *adjacent* if there is an edge between them (directed or undirected). For any node i the *neighborhood* $N(i)$ is the set of nodes adjacent to i and the *boundary* $\mathrm{Bd}(i)$ is the neighborhood of i including i: $\mathrm{Bd}(i) = N(i) \cup i$. We occasionally write $N_G(i)$ and $\mathrm{Bd}_G(i)$ to emphasize that we refer to graph G. The *degree* of a node i is the cardinality of $N(i)$. If there is an edge $i \to j$ we call i the *parent* of j and j the *child* of i. A *way* is a sequence $p_0, \ldots, p_k$ of nodes so that for all i, with $0 \leq i < k$, there is an edge connecting p_i and p_{i+1}. Such a sequence is called a *path* if $p_i \neq p_j$ holds for all $i, j \in [0, k]$ with $i \neq j$. A path from p_0 to p_k is called *causal* if every edge on the path is directed from p_i towards p_{i+1}. A node j is called an *ancestor* of i if there is a causal path from j to i. A node j is called a *descendant* of i if there is a causal path from i to j. $\mathrm{An}_G(i)$ is the set of all ancestors of i in graph G, $\mathrm{De}_G(i)$ is the set of all descendants of i in graph G. We use small letters for nodes and values, and capital letters for sets and random variables.

Of special importance are directed acyclic graphs (also called DAGs). As the name indicates these are directed graphs that do not contain a cycle. The *skeleton* of a DAG D is the undirected graph where every edge in D is substituted by an undirected edge. Another type of graphs we consider are partially directed acyclic graph (also called PDAGs). In these graphs undirected edges are allowed, however, directed cycles are not. A *topological sorting* of a DAG is an ordering of its nodes such that for every edge $i \to j$ the node i comes before j in the ordering. When speaking about a topological sorting for PDAGs we demand that for every *directed* edge $i \to j$ the node i comes before j in the ordering.

2.2 Probability notions

We will now associate the graphical models introduced above with a probability distribution. Let P be a joint probability distribution over the variables V and X, Y and Z stand

for any three sets of variables in V. We use the notation $(X \perp\!\!\!\perp Y \mid Z)$ to state that X is independent of Y given Z in P. Vice versa, $(X \not\perp\!\!\!\perp Y \mid Z)$ denotes that X and Y are dependent given Z. A DAG D is *compatible* with P if it factorizes P as $P(x_1, \ldots, x_n) = \prod_i (P(x_i \mid pa_i))$ where x_i is a particular realization of X_i and pa_i is a particular realization of the parents of X_i in D. Throughout this thesis we confine ourselves to independences of two variables given a conditioning set, meaning X and Y will usually be variables and not sets. Note that we use *iff* as shorthand for *if and only if*. It is possible to read CIs off a DAG through the notion of d-separation:

Definition 2.1 ([13])**.** *A path p is said to be* d-separated *(or* blocked*) by a set of nodes Z iff*

1. *p contains a chain $i \to m \to j$ or a fork $i \leftarrow m \to j$ such that the middle node m is in Z, or*

2. *p contains an inverted fork (or* collider*) $i \to m \leftarrow j$ such that the middle node m is not in Z and such that no descendant of m is in Z.*

A set Z is said to d-separate x from y iff Z blocks every path from x to y.

We write $(x \perp\!\!\!\perp y \mid Z)_D$ when x and y are d-separated by Z in the DAG D with x and y being nodes and Z a set of nodes. An inverted fork $i \to m \leftarrow j$ is a called an *unshielded collider* or *v-structure* if i and j are not adjacent. A *pattern* of a DAG D is the PDAG which has the identical adjacencies as D and which has an oriented edge $i \to j$ iff there is a vertex $k \notin N(i)$ such that $i \to j$ and $k \to j$ are in D. Essentially, in the pattern of D the only directed edges are the ones which are part of a v-structure in D.

A special case of PDAGs are the so called CPDAGs [2] or completed partially directed graphs. These graphs represent Markov equivalence classes. If two DAGs are Markov equivalent, it means that every probability distribution that is compatible with one of the DAGs is also compatible with the other [13]. As shown by Verma and Pearl [19] two DAGs are Markov equivalent *iff* they have the same skeleton and the same v-structures.

Definition 2.2. *Given a DAG $D = (V, E)$, the class of Markov equivalent graphs to D, denoted as $[D]$, is defined as $[D] = \{D' \mid D' \text{ is Markov equivalent to } D\}$. The graph representing $[D]$ is called a completed partially directed acyclic graph (CPDAG) and is denoted as $D^* = (V, E^*)$, with the set of edges defined as follows: $i \to j$ is in E^* if $i \to j$ belongs to every $D' \in [D]$ and $i - j$ is in E^* if there exists $D', D'' \in [D]$ so that $i \to j$ is an edge of D' and $i \leftarrow j$ is an edge of D''. A partially directed graph G is called a CPDAG if $G = D^*$ for some DAG D.*

A CPDAG represents a Markov equivalence class and every DAG in this class is a consistent extension of the CPDAG:

Definition 2.3. *Given a partially directed graph G, a DAG D is an extension of G iff G and D have the same skeleton and if $i \to j$ is in G, then $i \to j$ is in D. An extension is called* consistent *if additionally G and D have the same v-structures. We denote the set of all consistent extensions of G as $CE(G)$.*

Meek [12] proved the following theorem:

Theorem 2.1 ([12])**.** *When starting with a pattern of some DAG and repeatedly executing the following three rules until none of them applies, we obtain a CPDAG:*

R1: Orient $j - k$ into $j \to k$ whenever there is $i \to j$ such that i and k are nonadjacent.

R2: Orient $i - j$ into $i \to j$ whenever there is a chain $i \to k \to j$.

R3: Orient $i - j$ into $i \to j$ whenever there are two chains $i - k \to j$ and $i - l \to j$ such that k and l are nonadjacent.

We will call the rules R1-R3 the *Meek rules*. We note that this theorem does not only hold when starting with the pattern of a DAG, but more generally for any PDAG whose consistent extensions form a Markov equivalence class.

2.3 Definitions for sets of independences

The following new definitions are necessary to formally model the problem of causal structure learning from sets of low-order CIs. The set $\mathcal{I}_V$ consists of CIs over variables $V = \{x_1, \dots, x_n\}$. These CIs have the form $(a \perp\!\!\!\perp b \mid Z)$ with $a, b \in V$ and $Z \subseteq V$. In particular, we commonly consider sets which only contain CI statements with $|Z| \leq k$. We denote these sets as $\mathcal{I}_V^k$. We will term the cardinality of the conditioning set the *order*, so $\mathcal{I}_V^k$ would be a set of CI statements up to order k. The set $\mathcal{I}_V^0$ solely contains marginal independences. Having access to the conditional independence information through a set (or sometimes a list) $\mathcal{I}_V$ or $\mathcal{I}_V^k$ is the standard setting in the literature [12, 18, 20]. Whenever we measure the complexity of an algorithm, however, we will formally work with an oracle which tells us if a CI $(a \perp\!\!\!\perp b \mid Z)$ is in $\mathcal{I}_V$. A DAG D is called *faithful* to a set $\mathcal{I}_V^k$ if it contains exactly the independences of order $\leq k$ given in this set. A set $\mathcal{I}_V^k$ will be termed *DAG-representable* if there is a DAG which is faithful to it. We call a DAG D faithful to $\mathcal{I}_V^k$ *edge maximal* if there is no faithful DAG whose edge set is a superset of D. For a more elegant notation we usually write $(x \perp\!\!\!\perp y \mid Z)_{\mathcal{I}_V^k}$ instead of $(x \perp\!\!\!\perp y \mid Z) \in \mathcal{I}_V^k$ and vice versa $(x \not\perp\!\!\!\perp y \mid Z)_{\mathcal{I}_V^k}$ instead of $(x \perp\!\!\!\perp y \mid Z) \notin \mathcal{I}_V^k$. This notation is justified as we will only investigate DAG-representable sets of independences throughout this thesis. We will explicitly state that a set $\mathcal{I}_V^k$ is DAG-representable whenever we use this property in an argument. In all other cases the DAG-representability is implicitly assumed. For a set $\mathcal{I}_V^k$ we denote the set of all faithful DAGS as $\mathcal{F}(\mathcal{I}_V^k)$. We will say that a PDAG $G = (V_G, E_G)$ *contains* a set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^k)$ if it holds for every $D = (V_D, E_D) \in \mathcal{F}(\mathcal{I}_V^k)$ that $V_D = V_G$

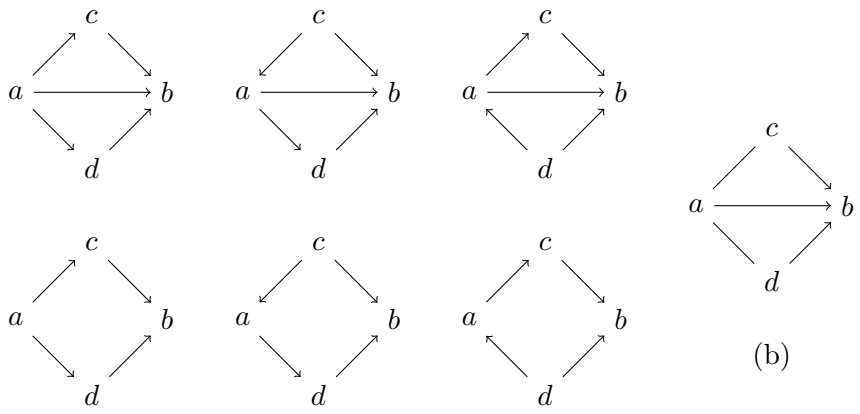

Figure 1: The set $\mathcal{F}(\mathcal{I}_V^1)$ for $V = \{a, b, c, d\}$ and $\mathcal{I}_V^1 = \{(c \perp\!\!\!\perp d \mid \{a\})\}$ is presented in (a) and its representation in (b).

and $E_D \subseteq E_G$. From a causal structure learning perspective it is our goal to extract as much information as possible from a set of CIs of order $\leq k$. We formalize this goal as the minimal PDAG which contains every DAG faithful to $\mathcal{I}_V^k$:

Definition 2.4. *A PDAG G* represents *the set $\mathcal{F}(\mathcal{I}_V^k)$ if G is a minimal graph that contains every graph in $\mathcal{F}(\mathcal{I}_V^k)$. Here, minimality is considered in regard to the inclusion relation between the sets of edges and taking into account that an undirected edge $a - b$ in G is encoded as $a \to b$ and $b \to a$.*

Later we will show that G is actually a CPDAG (see Corollary 6.2). We note that a PDAG G representing a set $\mathcal{F}(\mathcal{I}_V^k)$ fulfills the following conditions:

1. There is an edge $a - b$ in G iff DAGs $D_1, D_2 \in \mathcal{F}(\mathcal{I}_V^k)$ exist such that there is an edge $a \to b$ in D_1 and an edge $a \leftarrow b$ in D_2.

2. There is an edge $a \to b$ in G iff a DAG $D \in \mathcal{F}(\mathcal{I}_V^k)$ exists which contains the edge $a \to b$ and no DAG in $\mathcal{F}(\mathcal{I}_V^k)$ contains the edge $a \leftarrow b$.

3. There is no edge between a and b in G iff no DAG in $\mathcal{F}(\mathcal{I}_V^k)$ contains an edge between a and b.

From this perspective one can already view the representation G as a generalization of the notion of a CPDAG. Moreover, we will later show that the consistent extensions of G are faithful DAGs.

We will finish this section with an example which illustrates the new terms introduced above. This example is depicted in Figure 1. We have the set of variables $V = \{a, b, c, d\}$

and the set of independences $\mathcal{I}_V^1 = \{(c \perp\!\!\!\perp d \,|\, \{a\})\}$. Note that for the sake of readability we refrain from listing independences that follow symmetrically as, for example, $(d \perp\!\!\!\perp c \,|\, \{a\})\}$ would in this case. The set $\mathcal{I}_V^1$ contains all conditional independences with order ≤ 1. This set is DAG-representable because $\mathcal{F}(\mathcal{I}_V^1)$ is not empty. As shown in part (a) of Figure 1 there are six DAGs faithful to $\mathcal{I}_V^1$. We can see that not all DAGs have the same skeleton. The three DAGs in the upper row are edge maximal. In the lower row the edge $a \to b$ could be added. In part (b) the representation of $\mathcal{F}(\mathcal{I}_V^1)$ is shown. We will refer to this graph as G. This is the minimal graph which contains every faithful DAG. For example the edge $a - c$ is undirected, because there is a faithful DAG with the edge $a \to c$ and one with the edge $a \leftarrow c$. All those DAGs have to be contained in G and therefore the edges $a \to c$ as well as $a \leftarrow c$ are in G which, in turn, is encoded as an undirected edge. On the other hand, the directed edge $c \to b$ is present in G because there is no faithful DAG containing the edge $c \leftarrow b$ (while there is a DAG which contains $c \to b$). We will briefly explain why the edge $c \leftarrow b$ is in no faithful DAG. If this edge were in a faithful DAG, it would imply that there is an unblocked path between c and d given the conditioning set $Z = \{a\}$. This is because there can be no collider at node b with the edge $c \leftarrow b$ and $(b \not\perp\!\!\!\perp d \,|\, Z)_{\mathcal{I}_V^1}$. With $b \notin Z$ this would contradict $(c \perp\!\!\!\perp d \,|\, Z)_{\mathcal{I}_V^1}$. It is a bit harder to see why no faithful DAG contains the edge $a \leftarrow b$. We investigate this further in Section 6.

3 Previous work

In this section we summarize the most relevant work on the topics dealt with in this thesis. When it comes to the problem of recovering causal structure from a set of order-bounded independences, the special case of marginal independences is the one which has been studied the most intensively. But there are also some works which include conditional independences of order one. There are, however, no generalizations that consider a fixed k as the bound on the order. We will also give a short overview of the topic of causal structure learning from complete conditional independence information. In particular, we recall the popular PC algorithm.

3.1 Sets of marginal independences

Pearl and Wermuth [14] investigate the problem whether a set of marginal independences $\mathcal{I}_V^0$ has a causal interpretation — meaning a DAG faithful to $\mathcal{I}_V^0$. Moreover, they propose an algorithm to construct a faithful DAG (Algorithm 1 below). However, in their paper they do not give proofs for their theorems. Textor et al. [18] investigate the stated problem further, characterizing the DAG-representable sets by graph theoretical properties of the marginal independence graphs (these are undirected graphs with an edge between i and j iff i and j are marginally dependent). Additionally, they propose Algorithm 1 below which is based on above mentioned construction by Pearl and Wermuth [14]. We want to

input : Vertex set V, a DAG-representable set $\mathcal{I}_V^0$ of marginal independence statements
output: CPDAG H which contains every faithful DAG and whose extensions are faithful

1 Form the empty graph H on the vertex set V and the graph $\mathcal{U}$ which has an undirected edge $a - b$ if $(a \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$.
2 **foreach** *edge* $u - v$ *in* $\mathcal{U}$ **do**
3 | Add the edge $u \to v$ to H if $\mathrm{Bd}_{\mathcal{U}}(u) \subset \mathrm{Bd}_{\mathcal{U}}(v)$.
4 | Add the edge $u \leftarrow v$ to H if $\mathrm{Bd}_{\mathcal{U}}(u) \supset \mathrm{Bd}_{\mathcal{U}}(v)$.
5 | Add the edge $u - v$ to H if $\mathrm{Bd}_{\mathcal{U}}(u) = \mathrm{Bd}_{\mathcal{U}}(v)$.
6 **end**

Algorithm 1: Algorithm from Textor et al. to find faithful DAGs for sets of marginal independences [18].

emphasize that when adding edges between line 3 and 5 in Algorithm 1 an edge will not be added between u and v if neither $\mathrm{Bd}(u) \subseteq \mathrm{Bd}(v)$ nor $\mathrm{Bd}(u) \supseteq \mathrm{Bd}(v)$ hold. No proof is given for the statement that the extensions of H are indeed faithful to $\mathcal{I}_V^0$. Based on Algorithm 1 a method is proposed which enumerates every DAG faithful to $\mathcal{I}_V^0$. Moreover,

it is remarked that H is a CPDAG, but here as well no proof is given. It remains unclear how these results can be generalized for sets of conditional independences.

3.2 Independences with order ≤ 1

The problem of finding faithful DAGs for a set of marginal independences and conditional independences with a singleton conditioning set has not been extensively discussed in the literature yet. However, there are a few papers which touched on the topic. Campos and Huete [7] propose an algorithm for causal structure learning which uses zero- and first-order independences to improve on previous constraint-based-algorithms. They also access conditional independences of higher order, but the goal is to learn as much as possible from low-order independences. However, they presuppose knowledge of the topological ordering of the underlying DAG which is a rather strong assumption.

Wille and Bühlmann propose [21] to learn the undirected graph with those edges present for which no zero- or first-order independence exists. However, it is possible that this graph is not the skeleton for any faithful DAG. They analyze that for trees and other sparse DAGs, their approach is a good heuristic.

3.3 Complete conditional independence information

The question of finding the faithful DAGs when given complete information has been examined extensively beginning with Verma and Pearl [20]. Having complete independence information is comparable to the real world scenario of having access to CI tests. As these CI tests are computationally expensive the main goal for practical use is, thus, to minimize the number of needed CI tests. Moreover the statistical properties of these tests are relatively weak — in particular for large conditioning sets — making it desirable to work with sets which are as small as possible. A number of algorithms have been proposed. The most prominent one is the so called PC algorithm. It has the advantage of needing only a polynomial amount of CI tests if the maximal degree $d_{\max}$ in the underlying DAG is fixed. This algorithm was first proposed by Spirtes, Glymour and Scheines in Chapter 5 of [17]. Later Kalisch and Bühlmann popularized the algorithm by proving consistency for Gaussian distributions in high dimensional, sparse settings [8]. We present the algorithm as given by Kalisch and Bühlmann. The PC algorithm consists of two parts: First, the skeleton is learned by searching for independences as shown in Algorithm 2. Afterwards, the conditioning sets found in the first step are used to form the necessary v-structures. Finally, the PDAG is maximally extended by the Meek rules (R1-R3). This is presented in Algorithm 3. The result is a CPDAG which represents the class of Markov-equivalent faithful DAGs.

For a proof of correctness when having access to the true conditional independences (the setting we consider in this thesis), we refer the reader to [17]. The number of carried out CI tests is bounded by $n \cdot (n-1) \cdot \sum_{i=0}^{d_{\max}} \binom{n-1}{i}$ [17]. This is because for every ordered

input : Vertex Set V, Conditional Independence Information $\mathcal{I}_V$
output: Estimated skeleton G_{skel}, separation sets S (only needed when directing the skeleton afterwards)

Form the complete undirected graph G_{skel} on the vertex set V. Set $l = -1$.
repeat
  Set $l = l + 1$.
  repeat
    Select a (new) ordered pair of nodes i, j that are adjacent in G_{skel} such that $|N(i)\backslash\{j\}| \geq l$.
    repeat
      Choose (new) $Z \subseteq N(i)\backslash\{j\}$ with $|Z| = l$.
      if $(i \perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V}$ **then**
        Remove the edge $i - j$ from G_{skel}.
        Save Z in $S(i, j)$ and $S(j, i)$.
      end
    until *edge $i - j$ is deleted or all $Z \subseteq N(i)\backslash\{j\}$ with $|Z| = l$ have been chosen*
  until *all ordered pairs of adjacent variables i and j such that $|N(i)\backslash\{j\}| \geq l$ and $Z \subseteq N(i)\backslash\{j\}$ with $|Z| = l$ have been tested for conditional independence*
until *for each ordered pair of adjacent nodes i, j: $|N(i)\backslash\{j\}| < l$*

Algorithm 2: PC algorithm — Learning the skeleton [8]

input : Skeleton G_{skel}, separation sets S
output: CPDAG G representing the Markov equivalence class of faithful DAGs

Set $G_{\text{patt}} = G_{\text{skel}}$.
foreach *chain $i - k - j$ in G_{skel} with i and j nonadjacent* **do**
  if $k \notin S(i, j)$ **then**
    Replace $i - k - j$ in G_{patt} by $i \to k \leftarrow j$.
  end
end
Set $G = G_{\text{patt}}$.
repeat
  R1: In G orient $j - k$ into $j \to k$ whenever there is an arrow $i \to j$ such that i and k are nonadjacent.
  R2: In G orient $i - j$ into $i \to j$ whenever there is a chain $i \to k \to j$.
  R3: In G orient $i - j$ into $i \to j$ whenever there are two chains $i - k \to j$ and $i - l \to j$ such that k and l are nonadjacent.
until *No further edges can be oriented in G by application of R1-R3.*

Algorithm 3: PC algorithm — Extending the skeleton to a CPDAG [8]

pair of nodes we search through all sets of neighboring nodes up to cardinality $d_{\max}$. In the worst case no independence with a conditioning set smaller than $d_{\max}$ is found, and thus every node has $n-1$ neighbors until sets of size $d_{\max}$ are considered. We can estimate:

$$\begin{aligned} n \cdot (n-1) \cdot \sum_{i=0}^{d_{\max}} \binom{n-1}{i} &\leq n \cdot (n-1) \cdot \sum_{i=0}^{d_{\max}} \frac{(n-1)^i}{i!} \\ &= n \cdot (n-1) \cdot \sum_{i=0}^{d_{\max}} \frac{d_{\max}^i}{i!} (n-1/d_{\max})^i \\ &\leq n \cdot (n-1) \cdot e^{d_{\max}} (n-1/d_{\max})^{d_{\max}} \end{aligned}$$

It follows that in the worst case the number of CI tests is asymptotically in $\mathcal{O}(n^{2+d_{\max}})$ and thus polynomial for a fixed maximal degree $d_{\max}$. It is reasonable to assume that the worst case occurs very rarely in practice and that the algorithm will usually perform even better.

Almost all proposed modifications of the PC algorithm seek out to make the algorithm more robust and improve the quality of the results on real word-data [5, 9, 11]. There is only one notable paper by Abellán et al. [1] which proposes modifications to improve the running time of the PC algorithm in the ideal scenario of having complete and correct independence information. However, the experiments in this paper do not indicate that the performances of these modifications are significantly better than the performance of the PC algorithm.

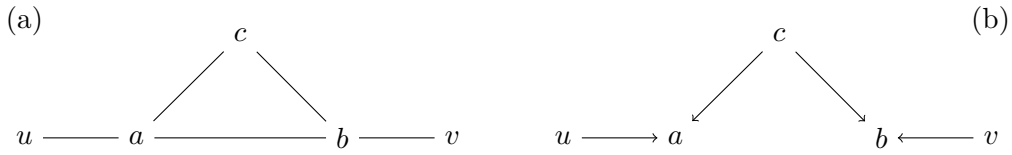

Figure 2: In (a) the set $\mathcal{I}_V^0 = \{(u \perp\!\!\!\perp c), (u \perp\!\!\!\perp b), (u \perp\!\!\!\perp v), (a \perp\!\!\!\perp v), (c \perp\!\!\!\perp v)\}$ for the vertex set $V = \{a, b, c, d, u, v\}$ is visualized by an undirected graph which contains an edge $a - b$ if a and b are marginally dependent. In (b) we show the only DAG faithful to $\mathcal{I}_V^0$.

4 Incompatible nodes

4.1 Motivation and Definition

In this thesis we are concerned with causal structure learning from sets of CIs. Of particular interest are such sets $\mathcal{I}_V^k$ where the order of the CIs is bounded by k. In order to recover the underlying causal structure our goal is to find the representation of the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^k)$. This is a PDAG (later we will show that the representation is actually a CPDAG) which contains every faithful DAG. Moreover, this PDAG should be minimal with regards to the edges. In other words, our goal is to remove all edges which are in no faithful DAG and, vice versa, keep all edges which are in at least one faithful DAG. In this section, we investigate how to find the edges which can be removed. This will also be useful in view of causal structure learning in general where the most common algorithms start with a complete graph and remove exactly the mentioned superfluous edges. Therefore, all definitions and theorems in this section apply to sets $\mathcal{I}_V$ as well as $\mathcal{I}_V^k$.

One way of removing edges is quite obvious. If we have a statement $(a \perp\!\!\!\perp b \,|\, Z)_{\mathcal{I}_V}$, it follows trivially that there cannot be an edge between a and b in any faithful DAG. Removing edges according to this rule is sufficient for causal structure learning when having access to the complete conditional independence information. For example, the PC algorithm works exactly in this fashion. However, only removing these edges is not sufficient for obtaining the skeleton of a faithful DAG when we consider *order-bounded* sets of independences. This can be exemplified by the following case: We have a set of marginal independences $\mathcal{I}_V^0 = \{(u \perp\!\!\!\perp c), (u \perp\!\!\!\perp b), (u \perp\!\!\!\perp v), (a \perp\!\!\!\perp v), (c \perp\!\!\!\perp v)\}$ for the vertex set $V = \{a, b, c, d, u, v\}$ whose dependences are shown as undirected edges in the graph in part (a) of Figure 2. It is immediately clear that any edge missing in this graph (for example $u - b$) means that u and b are nonadjacent in every faithful DAG. This is the case because these nodes need to be d-separated given the empty set with $(u \perp\!\!\!\perp b)$ being in $\mathcal{I}_V^0$. However, the nodes a and b also have to be nonadjacent in every DAG faithful to $\mathcal{I}_V^0$ even though we have $(a \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$. Essentially, this is the case because the edge between a and b (if present) would need to be in two conflicting v-structures namely $u \to a \leftarrow b$ and $a \to b \leftarrow v$ to make sure that u and b as well as a and v are marginally independent. This is clearly impossible. It follows

that there is no DAG faithful to $\mathcal{I}_V^0$ in which a and b are adjacent. Thus, it is vital that the nodes a and b are nonadjacent in the representation of $\mathcal{F}(\mathcal{I}_V^0)$ as well. Therefore, every algorithm learning this representation has to ensure that these edges are removed. We will formalize the situation just described in the following definition and thereby introduce the so called *incompatible nodes*:

Definition 4.1. *Given a set of CIs $\mathcal{I}_V$. Then two nodes a and b are called* incompatible *iff the following holds:*

$$(\exists u, Z)\ ((u \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V} \wedge (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{I}_V} \wedge (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V} \wedge a \notin Z)$$
$$\wedge\ (\exists v, Z')\ ((v \perp\!\!\!\perp a \mid Z')_{\mathcal{I}_V} \wedge (v \not\perp\!\!\!\perp b \mid Z')_{\mathcal{I}_V} \wedge (b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{I}_V} \wedge b \notin Z')$$

We can see that the nodes a and b in Figure 2 are incompatible because

$$(u \perp\!\!\!\perp b)_{\mathcal{I}_V^0},\ (u \not\perp\!\!\!\perp a)_{\mathcal{I}_V^0},\ (a \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0},\ (v \perp\!\!\!\perp a)_{\mathcal{I}_V^0} \text{ and } (v \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$$

hold. In this case Z and Z' are both the empty set. It follows immediately that $a \notin Z$ and $b \notin Z'$ hold. Moreover, $(b \not\perp\!\!\!\perp a)_{\mathcal{I}_V^0}$ follows by symmetry from $(a \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$ (and from the fact that $\mathcal{I}_V^0$ is DAG-representable).

4.2 Properties of incompatible nodes

We will now show the most important properties of incompatible nodes. We begin by proving that if the nodes a and b are incompatible, this means that there cannot be an edge between a and b in any faithful DAG.

Theorem 4.1. *Given a set of CIs $\mathcal{I}_V$. If we have $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V}$, $(u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{I}_V}$ and $a \notin Z$, then no DAG faithful to $\mathcal{I}_V$ contains the edge $a \to b$.*

Proof. Assume, for the sake of contradiction, there is an edge $a \to b$ in a faithful DAG D. In this DAG $(u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{I}_V}$ has to hold. This means that there is a path between u and a which is not blocked by Z. But as we have the edge $a \to b$ in G, there will also be a path between u and b which is not blocked by Z (note that $a \notin Z$). A contradiction. □

It immediately follows that incompatible nodes cannot be adjacent in any faithful DAG:

Corollary 4.1. *Given a set of CIs $\mathcal{I}_V$. If the nodes a and b are incompatible, they are nonadjacent in every DAG faithful to $\mathcal{I}_V$.*

Due to the conditions stated in the definition of incompatible nodes (Definition 4.1) it follows from Theorem 4.1 that neither the edge $a \to b$ nor $a \leftarrow b$ can be in any faithful DAG. We want to emphasize that (as can be seen in Theorem 4.1) the conditions $(a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V}$ and $(b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{I}_V}$ stated in Definition 4.1 are not necessary for a and b to be nonadjacent in every faithful DAG. If we would have $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V}$ or $(b \perp\!\!\!\perp a \mid Z')_{\mathcal{I}_V}$, trivially there

cannot be an edge connecting a and b. However, our point in Definition 4.1 is that the given conditions are nontrivial but still sufficient for two nodes to be nonadjacent in every faithful DAG. Therefore, we include $(a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ and $(b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V}$.

Theorem 4.1 showed that incompatible nodes cannot be adjacent in any faithful DAG. The following theorem will strengthen this statement further:

Theorem 4.2. *Given a set of CIs $\mathcal{J}_V$. If we have $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$, $(u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{J}_V}$ and $a \notin Z$, then no DAG faithful to $\mathcal{J}_V$ contains a causal path from a to b.*

Proof. Assume, for the sake of contradiction, that there is a faithful DAG D which contains a causal path from a to b even though the stated conditions hold. It follows from $(u \not\perp\!\!\!\perp a | Z)_{\mathcal{J}_V}$ and the faithfulness of D that there is a path from u to a which is not blocked by Z. But as u is supposed to be independent of b given Z, some node on the causal path from a to b (we call this node n) has to be in Z blocking this path. Moreover, we know that there is a path from a to b not blocked by Z as $(a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ has to hold. Thus, there has to be a collider at node a (because of $a \notin Z$) blocking this possible path from u to b. But this collider would be unblocked by node n as it is a successor of a and in Z. It follows that $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ does not hold. A contradiction. □

The following statement follows directly (as Corollary 4.1 above):

Corollary 4.2. *Assume $\mathcal{J}_V$ is a set of CIs. If the nodes a and b are incompatible, no DAG faithful to $\mathcal{J}_V$ contains a causal path between a and b.*

We have already seen an example of this fact in Figure 2. The nodes a and b are incompatible, but as a and b are marginally dependent there still is a path connecting them. However, this path is not causal but has the form $a \leftarrow c \rightarrow b$.

The notion of incompatible nodes is central in this thesis. When considering order-bounded sets of CIs $\mathcal{J}_V^k$, we will show that removing the possible edges between two incompatible nodes (in combination with all edges that we can remove trivially by having an independence in $\mathcal{J}_V^k$) is necessary and sufficient to obtain the skeleton of the representation of $\mathcal{F}(\mathcal{J}_V^k)$. The necessity of incompatible nodes being nonadjacent in the representation is clear from Figure 2. Proving the sufficiency on the other hand is much more complicate and will be one of the main results of this thesis.

We have seen the importance of the notion of incompatible nodes when considering order-bounded sets of CIs. But beyond that Definition 4.1 also applies to the setting of having complete conditional independence information without a bound on the order. In this case the removal of possible edges between incompatible nodes is not strictly necessary. With access to all CIs we are able to find a conditional independence for every pair of nonadjacent nodes in the underlying DAG. This is for example how the PC algorithm operates, by finding a separating set for every such pair. However, including the notion of incompatible nodes accelerates the learning of causal structures because we can infer independences earlier. This enables us to propose an improvement on the popular PC

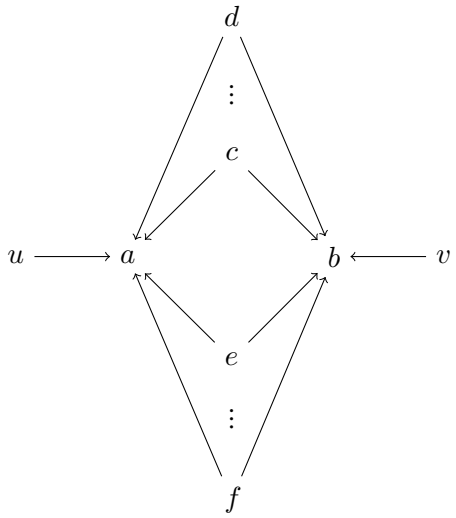

Figure 3: A class of DAGs for which a and b are separated by a large set $Z = \{c, \dots, d\} \cup \{e, \dots, f\}$. However, because the nodes a and b are incompatible, the edges $a \to b$ and $a \leftarrow b$ cannot be in any faithful DAG. This information can be retrieved already through marginal independences.

algorithm in Section 7. Thus, incompatible nodes play a key role in the standard setting of causal structure learning as well. We exemplify this in Figure 3.

A class of DAGs is shown for which the PC algorithm would need to consider the whole set $Z = \{c, \dots, d\} \cup \{e, \dots, f\}$ in order to separate a and b. However, finding this set is computationally very expensive. In particular, the PC algorithm will need at least $2^{|Z|}$ CI tests. But the nodes a and b are incompatible and it is clear that they cannot be adjacent in any faithful DAG as the following statements hold:

$$(u \perp\!\!\!\perp b), (u \not\perp\!\!\!\perp a), (a \not\perp\!\!\!\perp b), (v \perp\!\!\!\perp a), (v \not\perp\!\!\!\perp b).$$

Note that all of these statements are marginal independences and thus, we know that a and b are incompatible — and are able to remove the edges between a and b — just by looking at the marginal independences. Of course, what remains to be shown for a correct causal structure learning algorithm is that we are able to direct the edges $a - c - b$ into a v-structure if necessary. When removing the edges $a \to b$ and $a \leftarrow b$ because a and b are incompatible, we have to do this without knowing the separating set. This is nontrivial but indeed possible and shows that the notion of incompatible nodes is not simply an arbitrary condition, but instead a useful and important criterion for edge removal in causal structure learning. We show this result in Section 7 when we present an improved version of the PC algorithm.

However, a caveat is necessary. Even though it is possible to remove some edges much

earlier than the PC algorithm the number of CI tests will still grow polynomially in the maximal degree of the underlying DAG. In particular, in the example shown in Figure 3 the nodes a and b have a large degree and the algorithm still has to test for example whether u and a can be separated by any subset of $Z = \{c, \dots, d\} \cup \{e, \dots, f\}$ because all these nodes are neighbors of a. This limitation is inherent in constraint-based causal structure learning and all exact approaches have to check these CIs. Consider, for example, the extreme case that the skeleton of the underlying DAG is a complete graph. Then *every* algorithm learning this causal structure has to exhaustively check all possible CIs even though there is no conditional independence. We analyze this further in Section 7 and show experimentally in Section 8 that our proposed algorithm which removes edges between incompatible nodes is more efficient than the PC algorithms for most DAGs even with this limitation.

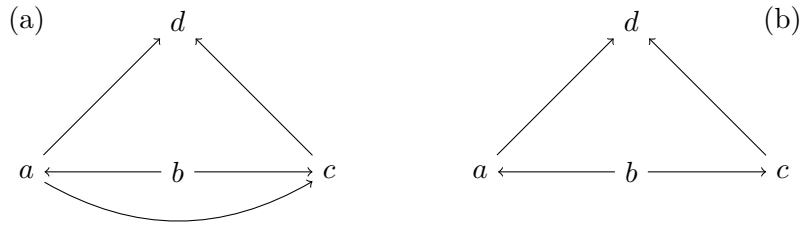

Figure 4: Both DAGs (a) and (b) have the same marginal independences (all variables are pairwise dependent), but a different skeleton and different v-structures.

5 Recovering causal structures from marginal independences

In this section our goal is to find the DAGs faithful to a set $\mathcal{I}_V^0$ of marginal independences. We do this by obtaining the representation G of the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^0)$. We will generalize this scenario to sets of CIs with a bounded order in the subsequent section. Still, the special case of marginal independences is of interest for two reasons. On the one hand there are some properties which only hold for the learning of causal structures from marginal independences. On the other hand this case has been investigated before by Pearl and Wermuth [14] as well as Textor et al. [18] and there are still some key proofs missing (see Section 3) which we prefer to give explicitly and not implicitly through the general case in the next section. Additionally, we want to give a simpler alternative to Algorithm 1 proposed by Textor et al. [18]. Finally, because the proofs in this section are considerably easier while the structure is similar to Section 6, the inclusion of this section emphasizes the new ideas we present.

5.1 Introduction of Algorithm 4

We are given a set of marginal independences $\mathcal{I}_V^0$. First, we investigate the structure of the set $\mathcal{F}(\mathcal{I}_V^0)$ of faithful DAGs. For Markov equivalent DAGs it holds that they have the same skeleton and the same set of v-structures. Neither of these two properties hold for DAGs faithful to a set of marginal independences, however, as is exemplified in Figure 4. In the DAG (b) the edge $a \to c$ is missing and the v-structure $a \to d \leftarrow c$ is present while in DAG (a) the edge $a \to c$ is present and therefore $a \to d \leftarrow c$ is no v-structure. But both DAGs have the same marginal independences as every pair of nodes is marginally dependent. This observation makes the recovery of the underlying causal structure much more challenging. In particular, our goal is first and foremost to find a minimal PDAG G which contains all faithful DAGs. We termed this PDAG the *representation* of the set of faithful DAGs (see Definition 2.4). When we obtain this representation we can be certain that we have extracted maximum information from the set $\mathcal{I}_V^0$ in a causal structure learning sense. More precisely, we know that every edge missing from G is not present in

the underlying DAG, while all remaining edges might be. Therefore, no further edges can be removed. We will later see that the representation also enables us to find the faithful DAGs. In particular, the consistent extensions of G are faithful to $\mathcal{I}_V^0$.

Algorithm 4 finds the representation G for a set of marginal independences $\mathcal{I}_V^0$. Proving this will be the main result of this section combined with the fact that G is a CPDAG. As mentioned above, in Section 6 we will propose a generalization (Algorithm 5) which recovers causal structures from a set of conditional independences with order $\leq k$.

input : Vertex set V, set $\mathcal{I}_V^0$ of marginal independence statements
output: CPDAG G which is the representation of $\mathcal{F}(\mathcal{I}_V^0)$
1 Form the graph $\mathcal{U}$ on the vertex set V which has an undirected edge $i - j$ if $(i \not\perp\!\!\!\perp j)_{\mathcal{I}_V^0}$. Set $G = \mathcal{U}$.
2 For every chain $i - k - j$ in $\mathcal{U}$ with i and j nonadjacent, remove the edges $i \leftarrow k$ and $k \rightarrow j$ from G.

Algorithm 4: Finding the representation G for a DAG-representable set $\mathcal{I}_V^0$ of marginal independences

Algorithm 4 consists of two steps: First, we form the graph that has (undirected) edges between marginally dependent nodes. Second, we remove all edges which cannot be in any faithful DAG because the conditions for Theorem 4.1 hold. This is the case for the edge $i \rightarrow k$ if $(k \not\perp\!\!\!\perp j)_{\mathcal{I}_V^0}$ and $(i \perp\!\!\!\perp j)_{\mathcal{I}_V^0}$ are satisfied. These conditions are met if we have the chain $i - k - j$ in $\mathcal{U}$. Thus, it is quite clear that the rules to remove edges from G are correct. In other words, these edges are in no faithful DAG. The main challenge is to show that every remaining edge in G is actually present in a faithful DAG. Then, it is clear that no further edge can be removed and it follows that G is the representation of $\mathcal{F}(\mathcal{I}_V^0)$.

We analyze the algorithm in more depth by considering the example which is shown in Figure 5. We have the set of marginal independences $\mathcal{I}_V^0 = \{(u \perp\!\!\!\perp c),\ (u \perp\!\!\!\perp b),\ (u \perp\!\!\!\perp v),\ (a \perp\!\!\!\perp v),\ (c \perp\!\!\!\perp v)\}$ with $V = \{a, b, c, d, u, v\}$. A DAG faithful to this set is shown in part (a) of Figure 5. The algorithm starts by producing the undirected graph $\mathcal{U}$ which contains an edge $a - b$ iff $(a \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$ holds. This graph is shown in part (b). Then we set $G = \mathcal{U}$ and obtain the resulting representation by removing edges from G according to the rule in line 2 of Algorithm 4. In this rule we remove the edges $i \leftarrow k$ and $k \rightarrow j$ from G if we have the chain $i - k - j$ with i and j nonadjacent in $\mathcal{U}$. It is important to recall how we formally model a graph in this thesis. Whenever we have the edges $a \rightarrow b$ and $a \leftarrow b$ in G, we encode this as an undirected edge $a - b$. This explains how we can remove the edge $a \leftarrow b$ from a graph with the edge $a - b$. This edge $a - b$ means that the edges $a \rightarrow b$ *and* $a \leftarrow b$ are present in the graph. Thus, if we remove the edge $a \leftarrow b$ from a graph G with the edge $a - b$, the edge $a \rightarrow b$ remains.

The graph shown in part (c) of Figure 5 is the graph G obtained after applying the rule in line 2 of Algorithm 4 to all chains starting with the node u. We show this intermediate

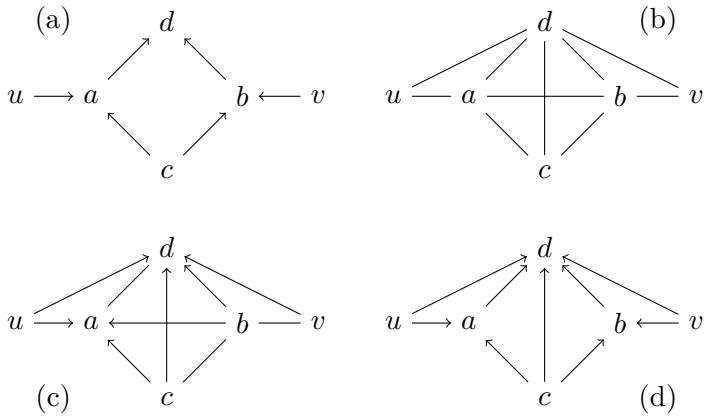

Figure 5: Example showing how Algorithm 4 works: The underlying DAG is shown in (a). In (b) we present the graph $\mathcal{U}$ obtained in line 1 of Algorithm 4. In (c) the intermediate graph G is shown after applying the rule in line 2 to all chains starting with node u. In (d) the final graph G is shown after applying the rule in line 2 to the remaining chains.

result in order to better illustrate how this rule works. We list all removed edges (some edges appear multiple times because they are in more than one chain): The edges $u \leftarrow d$ and $d \rightarrow v$ are removed because of the chain $u - d - v$ in $\mathcal{U}$, the edges $u \leftarrow a$ and $a \rightarrow c$ because of the chain $u - a - c$, the edges $u \leftarrow a$ and $a \rightarrow b$ because of the chain $u - a - b$ and the edges $u \leftarrow d$ and $d \rightarrow c$ because of the chain $u - d - c$.

In part (d) of Figure 5 the final result is shown. Here, all remaining chains are considered as well and the following edges are removed: The edges $c \leftarrow b$ and $b \rightarrow v$ are removed because of the chain $c - b - v$ in $\mathcal{U}$, the edges $a \leftarrow b$ and $b \rightarrow v$ because of the chain $a - b - v$ and the edges $a \leftarrow d$ and $d \rightarrow v$ because of the chain $a - d - v$. The resulting graph G is the representation of the set of faithful DAGs (we will prove this later). While G is actually a DAG in this example, this is not always the case and the representation might contain undirected edges. In general, G is a CPDAG as we will show in Theorem 5.4. We also want to emphasize that the nodes a and b are incompatible. They are nonadjacent in G even though we have $(a \not\perp\!\!\!\perp b)_{\mathcal{J}_V^0}$ because the conditions stated in Definition 4.1 are satisfied. More generally, all pairs of nodes that become nonadjacent during step 2 of Algorithm 4 are incompatible.

5.2 Equivalence of Algorithm 1 and Algorithm 4

In Section 3 we recalled a construction (see Algorithm 1) which was first proposed by Pearl and Wermuth [14] and later refined by Textor et al. [18]. This construction was used in order to find DAGs faithful to a set $\mathcal{J}_V^0$. We show that this algorithm is equivalent to Algorithm 4 we introduced in this section. The reason we propose a new algorithm for this

problem is that it is simpler and, most importantly, easier to generalize. Moreover, we are able to give the important proofs that every consistent extension of the resulting graph is faithful (Theorem 2 in [14] and Theorem 4.4 in [18]), that the result of the algorithm is the representation of $\mathcal{F}(\mathcal{I}_V^0)$ and that this representation is a CPDAG.

When comparing Algorithm 1 and Algorithm 4 it is clear that in both cases an edge will not be present in the resulting PDAG (we will only refer to the result of Algorithm 4 as a CPDAG after we have proven this in Theorem 5.4) if we have a statement $(a \perp\!\!\!\perp b)_{\mathcal{I}_V^0}$. What we will show is that the conditions for directing edges are equivalent as well. In particular both algorithms make sure that incompatible nodes are nonadjacent (see case 4 in the proof below).

Theorem 5.1. *For every set $\mathcal{I}_V^0$ of marginal independences Algorithm 4 produces the same PDAG as Algorithm 1.*

Proof. In this proof we denote the PDAG produced by Algorithm 1 as H and the one produced by Algorithm 4 as G. We have to show that $G = H$ holds. We will begin our proof by analyzing under which conditions a directed edge $u \leftarrow v$ is removed from G. This is the case if we have a node w which is a neighbor of v, but not a neighbor of u. Because then we have $u - v - w$ with u and w nonadjacent and in particular the edge $u \leftarrow v$ is removed.

Formally, the edge $u \leftarrow v$ (in case we have $(u \not\perp\!\!\!\perp v) \in \mathcal{I}_V^0$) is removed from G if the following condition holds:

$$(\exists w)(w \notin \mathrm{Bd}_{\mathcal{U}}(u) \wedge w \in \mathrm{Bd}_{\mathcal{U}}(v)) \tag{1}$$

Note that $\mathcal{U}$ is the same graph in both algorithms. We will also consider under which condition an edge $u \rightarrow v$ is not removed from G. This happens if condition 1 does not hold and by negation we get:

$$\begin{aligned} \neg((\exists w)(w \notin \mathrm{Bd}_{\mathcal{U}}(u) \wedge w \in \mathrm{Bd}_{\mathcal{U}}(v))) &= (\forall w)\neg(w \notin \mathrm{Bd}_{\mathcal{U}}(u) \wedge w \in \mathrm{Bd}_{\mathcal{U}}(v)) &(2)\\ &= (\forall w)(w \in \mathrm{Bd}_{\mathcal{U}}(v) \implies w \in \mathrm{Bd}_{\mathcal{U}}(u)) &(3)\end{aligned}$$

Now we can show that H and G are identical. We note that both PDAGs have the same vertex set. Thus, it is left to prove that all edges are identical. To do this we consider all possible edge states (undirected, directed or missing) between two node u and v in the following case study.

1. There is no edge between u and v in G and $(u \perp\!\!\!\perp v)_{\mathcal{I}_V^0}$ holds. Then, in the first line of both algorithms the edge was not added to $\mathcal{U}$ and thus is neither part of H nor G.

2. The directed edge $u \rightarrow v$ is in G. From above considerations it follows that the conditions
$$(\exists w)(w \notin \mathrm{Bd}_{\mathcal{U}}(u) \wedge w \in \mathrm{Bd}_{\mathcal{U}}(v))$$

and

$$(\forall w)(w \in \mathrm{Bd}_{\mathcal{U}}(u) \implies w \in \mathrm{Bd}_{\mathcal{U}}(v))$$

hold. This is because we require that the edge $u \leftarrow v$ was removed from G while $u \rightarrow v$ was not. Only then we have the directed edge $u \rightarrow v$ in G. Moreover, it is clear that the edge between u and v is present in $\mathcal{U}$ in both algorithms. We can see that the two conditions above are equivalent to $\mathrm{Bd}_{\mathcal{U}}(u) \subset \mathrm{Bd}_{\mathcal{U}}(v)$ which is exactly the condition in line 3 in Algorithm 1 for adding an edge $u \rightarrow v$ to H.

3. The directed edge $u \leftarrow v$ is in G. This case can be dealt with in the same way as $u \rightarrow v$ in case 2.

4. There is no edge between u and v in G and $(u \not\perp\!\!\!\perp v)_{\mathcal{J}^0_V}$. In Algorithm 4 this case occurs if the edges $u \rightarrow v$ and $u \leftarrow v$ are removed from G in different iterations in line 2. Thus, the following two conditions hold:

$$(\exists w)(w \notin \mathrm{Bd}_{\mathcal{U}}(u) \wedge w \in \mathrm{Bd}_{\mathcal{U}}(v))$$

and

$$(\exists x)(x \in \mathrm{Bd}_{\mathcal{U}}(u) \wedge x \notin \mathrm{Bd}_{\mathcal{U}}(v)).$$

This means that none of the three cases from line 3 to 5 in Algorithm 1 apply as $\mathrm{Bd}_{\mathcal{U}}(u)$ and $\mathrm{Bd}_{\mathcal{U}}(v)$ are not equal nor is one a subset of the other. This means that no edge is added to H. We note here that the opposite direction holds as well, meaning that if none of the three cases apply it follows that the two statements above concerning the existence of w and x are valid. Moreover, the nodes u and v are incompatible as

$$(w \perp\!\!\!\perp u)_{\mathcal{J}^0_V},\ (w \not\perp\!\!\!\perp v)_{\mathcal{J}^0_V},\ (v \not\perp\!\!\!\perp u)_{\mathcal{J}^0_V},\ (x \perp\!\!\!\perp v)_{\mathcal{J}^0_V} \text{ and } (x \not\perp\!\!\!\perp u)_{\mathcal{J}^0_V}$$

hold.

5. There is an edge $u - v$ in G. This can only occur in Algorithm 4 if neither $u \rightarrow v$ nor $u \leftarrow v$ get removed. But this means as reasoned above that

$$(\forall x)(x \in \mathrm{Bd}_{\mathcal{U}}(u) \implies x \in \mathrm{Bd}_{\mathcal{U}}(v))$$

and

$$(\forall x)(x \in \mathrm{Bd}_{\mathcal{U}}(v) \implies x \in \mathrm{Bd}_{\mathcal{U}}(u))$$

hold. It immediately follows that $\mathrm{Bd}_{\mathcal{U}}(u) = \mathrm{Bd}_{\mathcal{U}}(v)$ and therefore the edge $u - v$ is added to H in line 5 of Algorithm 1.

□

We have seen that the Algorithms 1 and 4 produce the same result. We will now compare the running time of the two algorithms. In order to simplify the analysis we measure the complexity in the cardinality of the vertex set n and will not consider the size of the set $\mathcal{I}_V^0$. Formally, this can be modeled by having access to an oracle which tells us if a certain marginal independence is in the set $\mathcal{I}_V^0$. In Algorithm 1 we have to compare the sets of neighbors for all adjacent nodes in $\mathcal{U}$. In the worst case there are $\mathcal{O}(n^2)$ adjacent nodes in $\mathcal{U}$ and each node can have $\mathcal{O}(n)$ neighbors. Thus, the worst case complexity is in $\mathcal{O}(n^3)$. In Algorithm 4 we consider all chains $i - k - j$ in $\mathcal{U}$. In the worst case there are $\mathcal{O}(n^3)$ of those chains and therefore the complexity is again in $\mathcal{O}(n^3)$. Thus, both algorithms have the same running time. As mentioned above, our main motivation for proposing Algorithm 1 was not to gain performance improvements, but that it is simpler and easier to generalize.

5.3 Proof of correctness

In this subsection we will prove the main results of this section that Algorithm 4 returns the representation G of the set $\mathcal{F}(\mathcal{I}_V^0)$ of faithful DAGs and that this representation is a CPDAG. Moreover, we show that every consistent extension of G is faithful. This proof has until now been missing from the literature (see Theorem 2 in [14] and Theorem 4.4 in [18]). During this whole subsection we assume that G is the result of applying Algorithm 4 to a set of marginal independences $\mathcal{I}_V^0$. The corresponding set of faithful DAGs is denoted as $\mathcal{F}(\mathcal{I}_V^0)$. We begin by stating that every faithful DAG is contained in G. This statement follows from Theorem 4.1 which shows the correctness of the rule for removing edges in line 2.

Lemma 5.1. *G contains every DAG $D \in \mathcal{F}(\mathcal{I}_V^0)$.*

Proof. Every faithful DAG is contained in the graph G formed in line 1. Moreover, by Theorem 4.1 every edge removed in line 2 is in no faithful DAG because $(a \perp\!\!\!\perp b)_{\mathcal{I}_V^0}$ and $(c \not\perp\!\!\!\perp b)_{\mathcal{I}_V^0}$ hold when the edge $a \leftarrow b$ is removed. □

What we need to show in order to prove that G represents the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^0)$ is the minimality of G. This is a much harder task. We begin by showing that every edge $a \to b$ in G can be added to a faithful DAG D producing another faithful DAG $D' = D \cup \{a \to b\}$ given that this does not produce a cycle. Proving this will be the first step towards showing that the edge maximal DAGs faithful to $\mathcal{I}_V^0$ have the same skeleton as G.

Theorem 5.2. *Given a DAG $D \in \mathcal{F}(\mathcal{I}_V^0)$ and two nodes $a, b \in V$ nonadjacent in D. The DAG $D' = D \cup \{a \to b\}$ is faithful to $\mathcal{I}_V^k$ iff $a \notin De_D(b)$ and $a \to b \in G$ hold.*

Proof. We show two directions. We begin by showing that if the DAG $D' = D \cup \{a \to b\}$ is faithful to $\mathcal{I}_V^0$, then $a \notin \mathrm{De}_D(b)$ and $a \to b \in G$ hold. Clearly, a cannot be in $\mathrm{De}_D(b)$

as then there would be a cycle in D'. Moreover, every faithful DAG is contained in G (Lemma 5.1) and because D' is faithful it follows that $a \to b$ is in G.

We will now show the more interesting direction that if $a \notin \mathrm{De}_D(b)$ and $a \to b \in G$ hold, the DAG $D' = D \cup \{a \to b\}$ is faithful to $\mathcal{I}^0_V$. We prove this by showing that the following holds:

$$\forall u, v \in V \quad [(u \perp\!\!\!\perp v)_{D'} \iff (u \perp\!\!\!\perp v)_D]$$

We show two directions. We begin with $(u \perp\!\!\!\perp v)_{D'} \implies (u \perp\!\!\!\perp v)_D$. It follows immediately from the fact that D is a subgraph of D' that every marginal independence in D' is also in D. The second direction $(u \not\perp\!\!\!\perp v)_{D'} \implies (u \not\perp\!\!\!\perp v)_D$ is more interesting. We consider a path p' which d-connects u and v in D'. We will show that there will also be a path p d-connecting u and v in D.

If the edge $a \to b$ is not part of p', this path also exists in D. If, on the other hand, $a \to b$ is in p' we will show that there is a path p_{u-b} d-connecting u and b in D. Then, as there will not be collider at node b (in p' the edge $a \to b$ was used and therefore the path between b and v has to start with $b \to$), a path p which d-connects u and v in D exists. This is the concatenation of p_{u-b} and the subpath p'_{b-v} between the nodes b and v of the path p'. Note that this concatenation is formally a way (because a node could be part of p_{u-b} and p'_{b-v}), but we know that every way d-connecting two nodes can be simplified into a path with the same property. We show now that the path p_{u-b} exists:

The statements $(u \not\perp\!\!\!\perp a)_{\mathcal{J}^0_V}$ and $(a \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ hold because of the validity of p'. It follows that $(u \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ holds as well. This is because in the case of $(u \perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ the edge $a \to b$ would have been removed in line 2 of Algorithm 4. A contradiction to the assumption that the edge $a \to b$ is in G. But with $(u \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ and the faithfulness of D it follows that there is a path p_{u-b} d-connecting u and b in the faithful DAG D and therefore as argued above also a path d-connecting u and v. □

The following Lemma will be important for the proof of Theorem 5.3 below.

Lemma 5.2. *If $a \leftarrow b \in G$ and $a \to b \notin G$, it follows that $b \notin De_D(a)$ holds for all DAGs $D \in \mathcal{F}(\mathcal{I}^0_V)$.*

Proof. Having the edge $a \leftarrow b$ in G but not the edge $a \to b$ implies that the following holds:

$$(\exists u \in V) \quad (u \perp\!\!\!\perp b)_{\mathcal{J}^0_V}, (u \not\perp\!\!\!\perp a)_{\mathcal{J}^0_V}, (a \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$$

These are the conditions that the edge $a \to b$ gets removed in line 2 of Algorithm 4. Note that this edge will not be removed in line 1 because $(a \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ holds. It follows from Theorem 4.2 that there cannot be a causal path from a to b in any DAG faithful to $\mathcal{I}^0_V$. Thus, b is not a descendant of a in all DAGs $D \in \mathcal{F}(\mathcal{I}^0_V)$. □

We prove the important result that the edge maximal faithful DAGs have the same skeleton as G. This shows that if two nodes a and b are nonadjacent in an edge maximal

faithful DAG either the independence $(a \perp\!\!\!\perp b)_{\mathcal{J}_V^0}$ holds or a and b are incompatible. This is because these are the only two ways a and b can become nonadjacent during the execution of Algorithm 4. In particular, edges between incompatible nodes are removed in line 2.

Theorem 5.3. *The edge maximal DAGs faithful to $\mathcal{J}_V^0$ have the same skeleton as G.*

Proof. We show two directions. If a and b are adjacent in an edge maximal faithful DAG D, they are also adjacent in G. This follows from Lemma 5.1 because every faithful DAG is contained in G.

The second direction is more intricate. We show that if a and b are adjacent in G, they are also adjacent in any edge maximal faithful DAG D. Assume, for the sake of contradiction, that a and b are not adjacent in an edge maximal faithful DAG D. We consider three cases:

1. The edges $a \to b$ and $a \leftarrow b$ are in G. From Theorem 5.2 we know that the edge $a \to b$ can be added to D if $a \notin \mathrm{De}_D(b)$. If on the other hand $a \in \mathrm{De}_D(b)$ holds, then the edge $a \leftarrow b$ can be added, because in this case $b \notin \mathrm{De}_D(a)$ has to hold (else there would be a cycle in D). Thus, D is not edge maximal. A contradiction.

2. The edge $a \leftarrow b$ is in G and the edge $a \to b$ is not. From Lemma 5.2 it follows that $b \notin \mathrm{De}_D(a)$ holds for every faithful DAG. Thus, as shown in Theorem 5.2 the edge $a \leftarrow b$ can be added to D. This means that D is not edge maximal. A contradiction.

3. The edge $a \to b$ is in G and the edge $a \leftarrow b$ is not. This case is symmetrical to case 2 above.

□

We have seen that all edge maximal faithful DAGs have the same skeleton and this is exactly the skeleton of G. Our goal is to show that the edge maximal DAGs faithful to $\mathcal{J}_V^0$ also have the same v-structures as G. We begin by proving the following Lemma:

Lemma 5.3. *Given a DAG-representable set $\mathcal{J}_V^0$ and a chain $a - c - b$ with a and b nonadjacent in the skeleton of G. Then the edges are directed either as $a \to c \leftarrow b$ or $a \leftarrow c \to b$ in G.*

Proof. The nodes a and b can be nonadjacent in G for two reasons:

1. The edge $a - b$ was not added to $\mathcal{U}$ in line 1 of Algorithm 4 because $(a \perp\!\!\!\perp b)_{\mathcal{J}_V^0}$ holds. Then the chain $a - c - b$ is in $\mathcal{U}$ and thus, considered in line 2. It follows that the edges $a \leftarrow c$ and $c \to b$ were removed and the chain is thus directed as $a \to c \leftarrow b$. Note that the edges $a \to c$ and $c \leftarrow b$ will be present in G because we assumed that the skeleton of G contains the chain $a - c - b$.

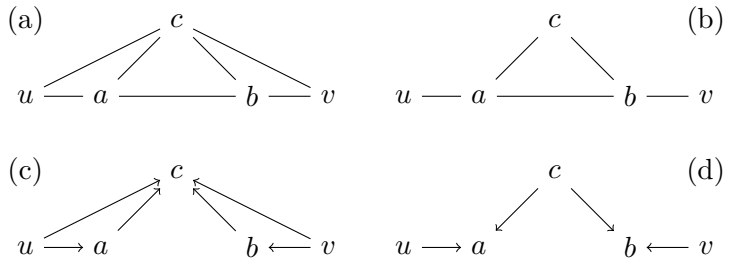

Figure 6: The two cases considered in the proof of Lemma 5.3. Case 2a is on the left in parts (a) and (c) and case 2b on the right in parts (b) and (d). In the upper row the graph $\mathcal{U}$ is shown which visualizes the marginal independences in $\mathcal{I}_V$. In the lower row the corresponding representation — meaning the learned structure — is shown.

2. The edges $a \to b$ and $a \leftarrow b$ were removed in line 2 of Algorithm 4. Then we have a node u with $(u \perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ and $(u \not\perp\!\!\!\perp a)_{\mathcal{J}^0_V}$ as well as a node v with $(v \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ and $(v \perp\!\!\!\perp a)_{\mathcal{J}^0_V}$. Now we consider the node c which is a common neighbor of a and b. In particular $(a \not\perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ and $(c \not\perp\!\!\!\perp b)_{\mathcal{J}^0_V}$ hold. There are two cases which are also illustrated in Figure 6. Our reasoning is that starting from the two possible statements $(u \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ and $(u \not\perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ due to the DAG-representability of $\mathcal{I}_V$ further statements can be deduced. In the end it becomes clear that these imply either $a \to c \leftarrow b$ or $a \leftarrow c \to b$ in the learned representation G.

   (a) If $(u \not\perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ holds the edge $c \to b$ will be removed from G in line 2 (we have the chain $u - c - b$ in $\mathcal{U}$). We know that c and b are adjacent and thus, it follows that the edge $c \leftarrow b$ is present in G. But then $(v \not\perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ has to hold as in case of $(v \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ the edge $c \leftarrow b$ would have been removed from G (we would have the chain $c - b - v$ in $\mathcal{U}$). This means, in turn, that we have $a \to c$ in G because as above the edge $a \leftarrow c$ gets removed due to the chain $a - c - v$ in $\mathcal{U}$. It follows that we get the v-structure $a \to c \leftarrow b$ in G.

   (b) If $(u \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ holds, it follows that we have $a \leftarrow c$ in G as the edge $a \to c$ is removed (due to the chain $u - a - c$ in $\mathcal{U}$) and as a and c are adjacent in G. Then $(v \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ holds because in case of $(v \not\perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ the edge $a \leftarrow c$ would have been removed because of the chain $a - c - v$ in $\mathcal{U}$. Thus, the edge $c \to b$ is removed from G because of the chain $c - b - v$ in $\mathcal{U}$ and we get the structure $a \leftarrow c \to b$.

   To summarize the two cases (a) and (b), for a DAG-representable set $\mathcal{I}_V$ the statement $(u \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ holds iff $(v \perp\!\!\!\perp c)_{\mathcal{J}^0_V}$ holds.

□

Corollary 5.1. *There is a v-structure in an edge maximal DAG $D \in \mathcal{F}(\mathcal{I}_V^0)$ iff it is present in G.*

Proof. We begin by showing that whenever there is a v-structure $a \to c \leftarrow b$ in D, it will also be present in G. From Theorem 5.3 we know that G has the same skeleton as D and thus, a and b are nonadjacent in G. It follows from Lemma 5.3 that the edges between a and c as well as c and b are already directed. Because D is contained in G (Lemma 5.1) we cannot have $a \leftarrow c \to b$ in G. Thus, we have the v-structure $a \to c \leftarrow b$ in G.

We show that a v-structure $a \to c \leftarrow b$ in G will also be in D. In particular, this means that the edges $a \leftarrow c$ and $c \to b$ were removed from G and it follows from Lemma 5.1 that they cannot be present in D either. Because D and G have the same skeleton, the v-structure $a \to c \leftarrow b$ will also be present in D. □

We have seen that all edge maximal faithful DAGs have the same skeleton and the same set of v-structures. Thus, these DAGs form a Markov-equivalence class:

Corollary 5.2. *The set of edge maximal DAGs faithful to $\mathcal{I}_V^0$ is the Markov equivalence class formed by all consistent extensions of G.*

Proof. Theorem 5.3 states that all edge maximal faithful DAGs have the same skeleton and Corollary 5.1 states that all edge maximal faithful DAGs have the same set of v-structures as G. Thus, the set $\mathcal{F}(\mathcal{I}_V^k)$ is equivalent to the set $CE(G)$ of consistent extensions of G. □

This finally proves the correctness of Theorem 2 in [14] and Theorem 4.4 in [18] which, essentially, stated that the consistent extensions of G are DAGs faithful to $\mathcal{I}_V^0$. It also shows that G is vital for finding faithful DAGs. Moreover, by connecting the set of edge-maximal faithful DAGs with G we can derive some interesting properties. In particular, we will show now that G is a CPDAG which enables us to conclude that G is minimal. The following Lemma shows that in G all transitive edges are present:

Lemma 5.4. *If we have the edges $a \to b \to c$ in the PDAG G obtained by Algorithm 4, there will also be an edge $a \to c$.*

Proof. We will first argue that an edge between a and c is present in G and then show that it is directed $a \to c$. Assume, for the sake of contradiction, that a and c are nonadjacent in G, but a and b are adjacent as well as b and c. Then from Lemma 5.3 we know that the edges are directed either as $a \to b \leftarrow c$ or $a \leftarrow b \to c$. A contradiction to above assumption that we have the edges $a \to b \to c$ in G.

For the sake of simplicity we will lead the following argument based on Algorithm 1 which, as we showed in Theorem 5.1, produces the same CPDAG as Algorithm 4. However, we note that it is possible to prove this statement solely with Algorithm 4 as well. From Algorithm 1 we know that $a \to b$ implies $\mathrm{Bd}_{\mathcal{U}}(a) \subset \mathrm{Bd}_{\mathcal{U}}(b)$ and $b \to c$ implies $\mathrm{Bd}_{\mathcal{U}}(b) \subset \mathrm{Bd}_{\mathcal{U}}(c)$. Thus, $\mathrm{Bd}_{\mathcal{U}}(a) \subset \mathrm{Bd}_{\mathcal{U}}(c)$ holds as well and the edge is directed as $a \to c$. □

Theorem 5.4. *G is a CPDAG.*

Proof. Meek proved that when starting with a pattern of a DAG and applying the rules R1-R3 (we termed these rules the *Meek rules*) repeatedly, the result will be a CPDAG [12]. We note here that it is not necessary to start with a pattern. A PDAG whose consistent extensions form a Markov equivalence class is sufficient. This is the case here as shown by Corollary 5.2. We show that none of the Meek rules can be applied to G, implying that it is already a CPDAG:

1. There is no structure $i \to j - k$ with i and k nonadjacent in G because in this case we either have $i \to j \leftarrow k$ or $i \leftarrow j \to k$ as shown in Lemma 5.3. Therefore, the first Meek rule does not apply.

2. We have shown in Lemma 5.4 that when we have $i \to j \to k$ there will already be an edge $i \to k$ in G. Therefore, the second Meek rule does not apply.

3. If k and l are nonadjacent the edges $i - k$ and $i - l$ would already be directed as shown in Lemma 5.3. Therefore, the third Meek rule does not apply.

It follows that G is a CPDAG. □

We have seen that G contains all faithful DAGs, that its consistent extensions are faithful and that it is a CPDAG. The final result of this section is that G is minimal and, thus, the representation of the set $\mathcal{F}(\mathcal{I}_V^0)$. This shows that from a constraint-based causal structure learning perspective we have extracted as much information as possible from G as no further edges can be removed.

Theorem 5.5. *G is the representation of the set $\mathcal{F}(\mathcal{I}_V^0)$.*

Proof. As shown in Lemma 5.1 G contains every DAG in $\mathcal{F}(\mathcal{I}_V^0)$. Moreover G is minimal, because removing another edge would immediately violate the condition that G contains every DAG in $\mathcal{F}(\mathcal{I}_V^0)$. This is because G is a CPDAG (Theorem 5.4) whose consistent extensions are the edge maximal faithful DAGs (Corollary 5.2). □

5.4 Enumerating all faithful DAGs

After obtaining the representation G of $\mathcal{F}(\mathcal{I}_V^0)$ the natural question arises how we can actually find (or even better enumerate) all faithful DAGs. In particular, the challenge is that not all faithful DAGs have the same skeleton and the same set of v-structures (we have seen this in Figure 4). Thus, well known approaches which enumerate all Markov equivalent DAGs fail [12]. This special problem has already been investigated by Textor et al. [18]. In Section 5 of this work they extensively study the problem using Algorithm 1 which is equivalent to Algorithm 4 (see Theorem 5.1). They propose an algorithm which solves the problem and enumerates all faithful DAGs using the representation G as well as the set of edge minimal faithful DAGs.

6 Recovering causal structures from order-bounded sets of independences

In this section our goal is to find the DAGs faithful to a set $\mathcal{I}_V^k$ of CIs up to order k. We achieve this by obtaining the representation G of the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^k)$. We propose Algorithm 5 to solve this task. This algorithm is a generalization of Algorithm 4 from the previous section. When proving the correctness, we moreover show that the representation G is a CPDAG and that all consistent extensions of G are DAGs faithful to $\mathcal{I}_V^k$. Lastly, we investigate further how the representation enables us to find *all* faithful DAGs.

6.1 Introduction of Algorithm 5

input : Vertex Set V, DAG-representable set $\mathcal{I}_V^k$ of conditional independence statements with order $\leq k$
output: CPDAG G which is the representation of $\mathcal{F}(\mathcal{I}_V^k)$

Form the graph G_{ep} on the vertex set V which has an undirected edge $i - j$ if it holds for every subset Z of V that $(i \not\perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V^k}$.
foreach *three nodes i, k, j and every conditioning set Z* **do**
  if $(i \perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V^k}$, $(i \not\perp\!\!\!\perp k \mid Z)_{\mathcal{I}_V^k}$, $(k \not\perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V^k}$ *and* $k \notin Z$ **then**
    Remove the edges $i \leftarrow k$ and $k \rightarrow j$ from G_{ep}.
  end
end
Set $G = G_{\text{ep}}$.
repeat
  R1: In G orient $j - k$ into $j \rightarrow k$ whenever there is an arrow $i \rightarrow j$ such that i and k are nonadjacent.
  R2: In G orient $i - j$ into $i \rightarrow j$ whenever there is a chain $i \rightarrow k \rightarrow j$.
  R3: In G orient $i - j$ into $i \rightarrow j$ whenever there are two chains $i - k \rightarrow j$ and $i - l \rightarrow j$ such that k and l are nonadjacent.
until *No further edges can be oriented in G by application of R1-R3.*

Algorithm 5: Obtaining the representation G for a DAG-representable set of CIs up to order k

We begin by explaining how Algorithm 4 works. At first, the graph G_{ep} is formed which contains an undirected edge $i - j$ if there is no set $Z \subseteq V$ with $|Z| \leq k$ such that the CI $(i \perp\!\!\!\perp j \mid Z)$ is in $\mathcal{I}_V^k$. This undirected edge encodes the edges $i \rightarrow j$ and $i \leftarrow j$ which are both in G_{ep} at the beginning of the algorithm. Recall that the representation is obtained by removing all edges which are in no faithful DAG. Thus, edges are removed

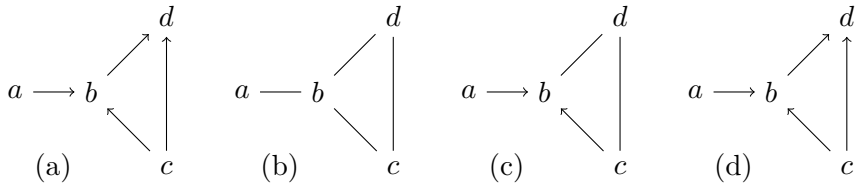

Figure 7: The underlying DAG faithful to the set $\mathcal{I}_V^1 = \{(a \perp\!\!\!\perp c), (a \perp\!\!\!\perp d \mid \{b, c\})\}$ with $V = \{a, b, c, d\}$ is shown in (a). In (b) the graph formed in line 1 is displayed. In (c) G_{ep} is shown after the for-loop from line 2 to 6. In (d) the final result G is shown after applying the Meek rules.

from G_{ep} according to the conditions stated in line 3. These are the conditions demanded in Theorem 4.1. We check exhaustively if the conditions are satisfied for every possible three nodes i, k, j and conditioning set Z. The result is a graph which we will call an extended pattern G_{ep}. This is due to the fact that all necessary v-structures are oriented (the usual definition of a pattern), but there are also directed edges which are not part of a v-structure. This notion of an extended pattern will be illustrated in more detail in the subsequent section.

Then, we set $G = G_{\text{ep}}$ and apply the Meek rules (R1-R3) to the graph G instead of G_{ep}. We do this, on the one hand, to emphasize the important intermediate result G_{ep} and, on the other hand, because we specifically need the graph G_{ep} for some proofs. After the application of the Meek rules we obtain the resulting graph G which is the representation of the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^k)$. We want to emphasize that there are two different ways two nodes a and b can be nonadjacent in G. The first is that there is a CI $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V^k}$. The second is that the edges $a \to b$ and $a \leftarrow b$ are removed in different iterations of the for loop between lines 2 and 6. If this second case applies the nodes a and b are incompatible. We show later in this section that these two rules for edges removals are sufficient in order to obtain the representation G of $\mathcal{F}(\mathcal{I}_V^k)$. Moreover, we will see that the consistent extensions of G are faithful DAGs.

Through the example depicted in Figure 7 we show in more detail how Algorithm 5 works. We have a set $\mathcal{I}_V^1 = \{(a \perp\!\!\!\perp c), (a \perp\!\!\!\perp d \mid \{b, c\})\}$ of CIs up to order 1. We consider the vertex set $V = \{a, b, c, d\}$. The underlying DAG is shown in part (a). The main goal of this example is to show why the Meek rules are needed in order to find the representation $\mathcal{F}(\mathcal{I}_V^k)$. These rules (R1-R3) were not necessary in the case of marginal independences we investigated in the previous section because the result was already a CPDAG (see Theorem 5.4).

The algorithm starts with the undirected graph G_{ep} in which two nodes are adjacent if there is no CI statement in $\mathcal{I}_V^1$ for those nodes. With $(a \perp\!\!\!\perp c)_{\mathcal{I}_V^1}$ and $(a \perp\!\!\!\perp d \mid \{b, c\})_{\mathcal{I}_V^1}$ the nodes a and c as well as a and d are therefore nonadjacent in G_{ep} while all other nodes are adjacent. The graph obtained in line 1 is shown in part (b) of Figure 7. In the for-loop

from line 2 to 6 edges are removed if the conditions for Theorem 4.1 apply. In this case the edges $a \leftarrow b$ and $b \rightarrow c$ are removed as we have $(a \not\perp\!\!\!\perp c)_{\mathcal{J}^1_V}$, $(a \not\perp\!\!\!\perp b)_{\mathcal{J}^1_V}$, $(b \not\perp\!\!\!\perp c)_{\mathcal{J}^1_V}$ and $b \notin \emptyset$. We have obtained the extended pattern G_{ep} which is shown in part (c).

We continue by applying the Meek rules to the graph $G = G_{\text{ep}}$. Through the first Meek rule the edge $b - d$ is oriented as $b \rightarrow d$ because we have $a \rightarrow b - d$ in G with a and d nonadjacent. This is correct because the opposite orientation $b \leftarrow d$ would imply a new v-structure. This, however, is not allowed because all necessary v-structures are already present in G_{ep}. Moreover, because of the chain $c \rightarrow b \rightarrow d$ the edge $c - d$ is oriented into $c \rightarrow d$ according to the second Meek rule. Clearly, the opposite orientation would create a cycle. The resulting representation is in this case equal to the underlying DAG which actually is the only DAG faithful to $\mathcal{J}^1_V$.

We have seen an application of the first two Meek rules. An example for the third rule is shown in Figure 1 we looked at in Section 2. There the edge $a - b$ is oriented as $a \rightarrow b$. This is correct as the orientation $a \leftarrow b$ would imply either a cycle or a new v-structure $c \rightarrow a \leftarrow d$. It is clear that the Meek rules are correct for a fixed skeleton. However, as we have seen already for marginal independences, the DAGs faithful to $\mathcal{J}^k_V$ not necessarily have the same skeleton. This makes it much harder to prove the correctness of the Meek rules in our setting. Take for example the second rule. Orienting the edge $i \rightarrow j$ avoids a cycle in G, but there might be a faithful DAG not containing the edge $i \rightarrow k$ or not containing the edge $k \rightarrow j$. It is necessary to show that in this DAG as well, the edge $i \leftarrow j$ will not be present. This is precisely the reason why we prefer to prove some statements for the graph G_{ep} and only afterwards include the Meek rules in our reasoning.

6.2 Comparing Algorithm 5 and Algorithm 4

Algorithm 4 which we introduced in the previous section solved the problem of recovering causal structures for the special case of marginal independences. This setting had already been investigated in the literature [14, 18] and, in particular, the equivalent Algorithm 1 (see Theorem 5.1) had been proposed. However, a generalization of this algorithm to the general setting we consider in this section was not possible and the problem remained unsolved. We will show that such a generalization is possible when instead starting with Algorithm 4 we proposed in the previous section. We prove in the following Theorem that Algorithm 5 (which finds the representation for the general problem as we show later in this section) generalizes Algorithm 4. This proof will also serve as a short comparison of the two algorithms.

Theorem 6.1. *Algorithm 5 produces the same CPDAG as Algorithm 4 when given a DAG-representable set $\mathcal{J}^0_V$ of marginal independences.*

Proof. Line 1 in Algorithm 5 is clearly equivalent to line 1 in Algorithm 4. Moreover the for-loop from line 2 to 6 in Algorithm 5 is equivalent to line 2 in Algorithm 4. This follows from the fact that the only conditioning set Z to consider is the empty set and that

$(i \perp\!\!\!\perp j)_{\mathcal{J}^0_V}$, $(i \not\perp\!\!\!\perp k)_{\mathcal{J}^0_V}$ and $(k \not\perp\!\!\!\perp j)_{\mathcal{J}^0_V}$ hold iff there is a chain $i - k - j$ in $\mathcal{U}$ with i and j nonadjacent. Finally, the Meek rules (R1-R3) will not orient any further edges because we have shown in Theorem 5.4 that in the case of marginal independences G is already a CPDAG. □

We have seen the two main differences of Algorithm 5 and Algorithm 4. First, that we have to consider every Z with $|Z| \leq k$ and second, that we have to include the Meek rules (R1-R3). It will become clear during the course of this section why this is necessary.

The running time of the algorithm is dominated by the for loop from line 2 to 6 and is, thus, in $\mathcal{O}(n^3 \cdot \sum_{i=0}^{k} \binom{n}{i})$. As in the previous section, we measure the complexity in the cardinality of the vertex set n and do not consider the size of $\mathcal{I}^k_V$. Formally, this is modeled by an oracle giving us access to this set. We can see that Algorithm 5 is asymptotically no worse than Algorithm 4 for the special case of $k = 0$. But practically speaking Algorithm 4 is faster because we only consider every chain $i - k - j$ in $\mathcal{U}$ instead of every three nodes $i, k, j \in V$. It is an open question if it is sufficient to only consider such chains in the general case as well. However, it is not our main goal to optimize the running time of Algorithm 5.

Instead when formulating Algorithm 5 our main interest is that we are able to prove the correctness as well as some other important properties (for example that G is a CPDAG). Therefore, it is our objective to present the algorithm as clean and simple as possible. The running time is only of secondary interest. In particular, this is because we will present a much more practicable algorithm (for the topic of causal structure learning in general) in the subsequent section. Still, we want to mention one further potential improvement. It might actually be sufficient to look at only one set Z per independence. This means that if we find one set for which $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}$ holds, it might not be necessary to look at another set Z' with $(a \perp\!\!\!\perp b \mid Z')_{\mathcal{J}^k_V}$ in the for loop from line 2 to 6.

6.3 Proof of correctness

In this subsection we show that the result of Algorithm 5 is a CPDAG which is the representation G of the set of faithful DAGs $\mathcal{F}(\mathcal{I}^k_V)$. Then it follows that no further edges can be removed from G. Recall that every edge in the representation is contained in a faithful DAG. Therefore, removing another edge from the representation G would mean that one or more DAGs will not be contained in G. At first, we will consider the PDAG obtained from Algorithm 5 at line 6 (after the for loop) before applying the Meek rules (R1-R3). Throughout this whole section we will refer to this PDAG as G_{ep} while we will refer to the output graph of Algorithm 5 as G. As explained above, considering G_{ep} instead of G will simplify some proofs and we will show the correctness of the three rules R1-R3 afterwards.

The structure of this subsection is very similar to the corresponding one in the previous section. However, a few technical proofs are much more intricate and may be skipped at first reading. In particular, this holds for Lemma 6.2 and Theorem 6.2. We begin by showing that every DAG faithful to a set of independences $\mathcal{I}^k_V$ is a subgraph of G_{ep}.

Lemma 6.1. *G_{ep} contains every DAG in $\mathcal{F}(\mathcal{J}^k_V)$.*

Proof. Every faithful DAG is contained in the graph formed in line 1. Moreover, for every edge $a \to b$ removed in line 4 the following holds:

$$\exists Z \subseteq V \text{ with } |Z| \leq k \quad \exists u \in V \quad [(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}, (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{J}^k_V}, (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}, a \notin Z]$$

By Theorem 4.1 the edge $a \to b$ is not part of any faithful DAG. □

Our goal is to show that the edge maximal DAGs have the same skeleton as G_{ep}. We begin with the following technical Lemma which is necessary for the proof of Theorem 6.2 below.

Lemma 6.2. *Given a DAG $D \in \mathcal{F}(\mathcal{J}^k_V)$, two nodes $a, b \in V$ and a set Z with $|Z| \leq k$ such that the following holds: $(a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}$, $(a \not\perp\!\!\!\perp b \mid Z \setminus De(b))_{\mathcal{J}^k_V}$ and $a \notin De(b)$. Then there is a path d-connecting a and b in D given Z which ends with $\to b$.*

Proof. Assume, for the sake of contradiction, that there is no path d-connecting a and b given Z ending with $\to b$ in D. This means every path ends with the edge $\leftarrow b$. Moreover we know that there cannot be a causal path from b to a because $a \notin \text{De}(b)$. Then, it is clear that every path p which d-connects a and b given Z in D contains at least one collider. We also note that on p every node unblocking the collider closest to b is a descendant of b. We will now consider the set $Z' = Z \setminus \text{De}(b)$ meaning we remove all nodes from Z which are a descendant of b. We will show that there can be no path d-connecting a and b given Z'. This will contradict the assumption that $(a \not\perp\!\!\!\perp b \mid Z')_{\mathcal{J}^k_V}$ holds for $Z' = Z \setminus \text{De}(b)$.

Every path d-connecting a and b given Z' contains a node $x \in \text{De}(b)$. If this were not the case and there actually is such a path which contains no node in $\text{De}(b)$, then this path would d-connect a and b given Z as well. Moreover, this path would have to end with the edge $\to b$ (else the node preceding b on the path is a descendant of b). But we have assumed above, for the sake of contradiction, that there is no path d-connecting a and b given Z in D ending with $\to b$.

We will consider a path p' d-connecting a and b given Z' which contains a node $x \in \text{De}(b)$. This node cannot be a collider $\to x \leftarrow$ in p', because x is not in Z' and neither is any descendant y of x as y is by transitivity a descendant of b as well. Thus, the collider $\to x \leftarrow$ would be unblocked. It follows that in p' there is an edge $\leftarrow x$ or an edge $x \to$. We investigate these two cases which are also displayed in Figure 8:

1. Consider the edge $\leftarrow x$ is in p'. This case is shown in part (a) of Figure 8. We denote the subpath between a and x of p' as p'_{a-x}. This subpath cannot be causal from x to a as then there would be a causal path from b to a because x is a descendant of b. But we required that $a \notin \text{De}(b)$ holds.

   This means that there has to be a collider on p'_{a-x}. We will look at the collider c_1 closest to x. The collider c_1, however, cannot be unblocked by a node d_1 in Z'. This is because d_1 would be a descendant of b.

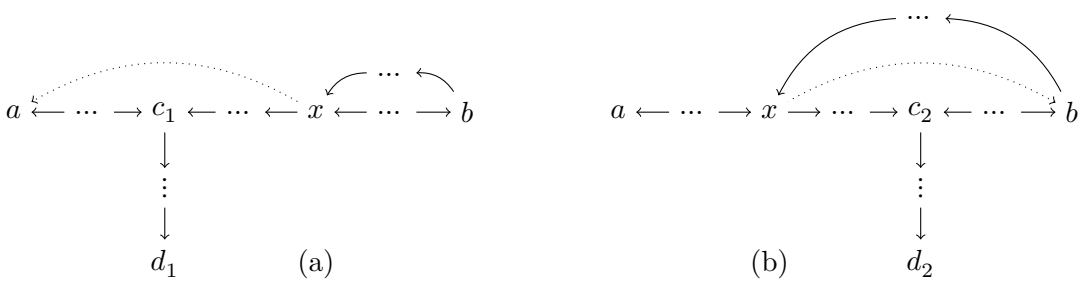

Figure 8: The two cases considered in the proof of Lemma 6.2. In (a) there is an edge $\leftarrow x$ on the path between a and b. A causal path from x to a (dotted line) is impossible because then there would be a causal path from b to a. We show that the collider c_1 is unblocked. In (b) the edge $x \rightarrow$ is part of the path between a and b. A causal path from x to b (dotted line) is impossible because this would imply a cycle. We show that the collider c_2 is unblocked.

2. Consider the edge $x \rightarrow$ is in p'. This case is shown in part (b) of Figure 8. We denote the subpath between x and b of p' as p'_{x-b}. This subpath cannot be causal from x to b as then there would be a cycle because x is a descendant of b.

   It follows that there is a collider on the subpath p'_{x-b}. We look at the collider c_2 closest to x. This collider cannot be unblocked by a node d_2 in Z' because d_2 would be a descendant of b.

We have seen that there cannot be a path d-connecting a and b given Z' in D. This is a contradiction to the requirement that $(a \not\perp\!\!\!\perp b \mid Z')_{\mathcal{J}^k_V}$ holds for $Z' = Z \setminus \mathrm{De}(b)$. Therefore, we conclude that indeed there is a path d-connecting a and b given Z ending with $\rightarrow b$ in D. □

The following theorem is of central importance for this section. We show that every edge in G_{ep} can be added to a faithful DAG D iff this does not produce a cycle. This is an important step towards showing that the edge maximal DAGs faithful to $\mathcal{J}^k_V$ have the same skeleton as G_{ep}.

Theorem 6.2. *Given a DAG $D \in \mathcal{F}(\mathcal{J}^k_V)$ and $a, b \in V$ nonadjacent in D. The DAG $D' = D \cup \{a \rightarrow b\}$ is faithful to $\mathcal{J}^k_V$ iff $a \notin De_D(b)$ and $a \rightarrow b \in G_{ep}$ hold.*

Proof. We show two directions. We begin by showing that if the DAG $D' = D \cup \{a \rightarrow b\}$ is faithful to $\mathcal{J}^k_V$, then $a \notin \mathrm{De}_D(b)$ and $a \rightarrow b \in G_{\text{ep}}$ hold. Clearly, a cannot be in $\mathrm{De}_D(b)$ as then there would be a cycle in D'. Moreover, every faithful DAG is contained in G_{ep} (Lemma 6.1) and because D' is faithful it follows that $a \rightarrow b$ is in G_{ep}.

We will now show the more interesting direction that if $a \notin \mathrm{De}_D(b)$ and $a \rightarrow b \in G_{\text{ep}}$ are satisfied, the DAG $D' = D \cup \{a \rightarrow b\}$ is faithful to $\mathcal{J}^k_V$. We prove this by showing that

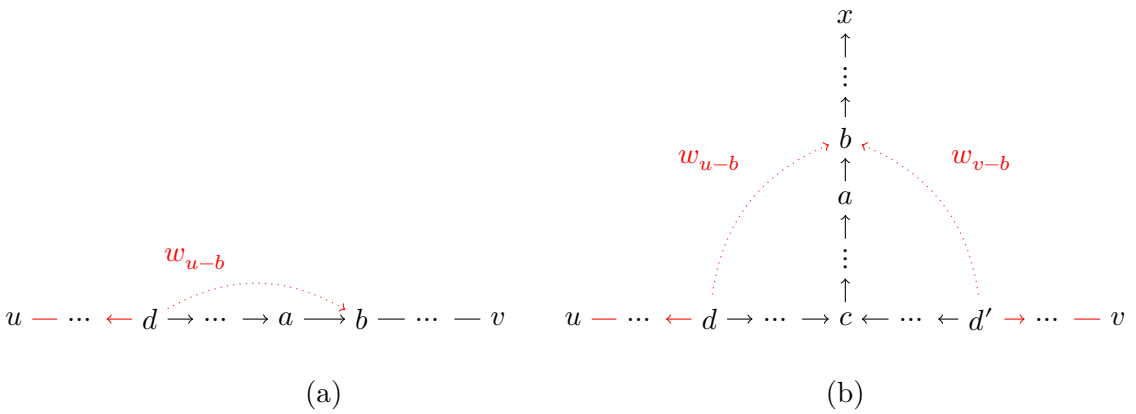

Figure 9: The two cases in which the edge $a \to b$ can be part of a path p' d-connecting u and v given Z in D'. We show that in both there is a path p d-connecting u and v given Z in D. In (a) the edge $a \to b$ is on the path p'. Then we can find a way w_{u-b} (indicated by the red arrows) d-connecting u and b given Z in D which ends with $\to b$. Combined with the existing path between b and v, we conclude that the desired path p exists. In (b) the edge $a \to b$ is part of a chain which unblocks a collider c on the path p'. We find a way w_{u-b} and by symmetry a way w_{v-b} (both indicated by the red arrows) which combined imply that the desired path p exists as the collider at node b will be unblocked by x.

the following holds:

$$\forall Z \subseteq V \text{ with } |Z| \leq k \quad \forall u, v \in V \quad [(u \perp\!\!\!\perp v \mid Z)_{D'} \iff (u \perp\!\!\!\perp v \mid Z)_{D}]$$

We show two directions. We begin with the direction $(u \perp\!\!\!\perp v \mid Z)_{D'} \implies (u \perp\!\!\!\perp v \mid Z)_{D}$. Every conditional independence of order $\leq k$ in D' is also in D because D is a subgraph of D'. The second direction $(u \not\perp\!\!\!\perp v \mid Z)_{D'} \implies (u \not\perp\!\!\!\perp v \mid Z)_{D}$ is more intricate. We will prove that every conditional dependence of order $\leq k$ in D' is also in D by considering a path p' which d-connects u and v given a set Z in D'. Then we show that there will also be a path p in D not blocked by Z.

There are two cases to consider displayed in Figure 9. The case (a) describes the situation when the edge $a \to b$ is on the path p' d-connecting u and v given Z in D'. The case (b) appears when a collider c on the path p' is unblocked by the descendant x which is in Z and the edge $a \to b$ is on the causal path from c to x. The nodes d and d' will be important later in the proof and can be ignored for now. It is clear that any further occurrence of the edge $a \to b$ in p' would be redundant. Moreover it is obvious that if none of the two cases applies and the edge is neither present in p' nor takes part in unblocking a collider, the same path p' will also exist in D.

We prove that for the two cases in Figure 9 there is a path p d-connecting u and v given Z in D. We do this by showing that a way w_{u-b} between u and b exists which

does not contain the edge $a \to b$, but is still ending with $\to b$. It is necessary that this way ends with $\to b$ (as the subpath p'_{u-b} of p' does) because b could be a collider which is unblocked by b or a descendant of b being in Z. Interestingly, we did not have to consider this case for marginal independences (see the proof of Theorem 5.2) because when Z is the empty set, no colliders can be unblocked. This is precisely what makes the proof harder in the general case of conditional independences. We will now show that if a way w_{u-b} exists which d-connects u and b given Z, then there will be a path p d-connecting u and v given Z in D.

In both cases of Figure 9 there exists such a path p not containing the edge $a \to b$. In case (a) we have a way w which is the concatenation of the way w_{u-b} (this path is indicated by the red arrows) and the path p'_{b-v} which is the subpath between b and v of path p'. In case (b) the way w is the concatenation of the way w_{u-b} and w_{v-b} (this way exists by symmetry). Both ways are indicated by the red arrows and we will introduce the nodes d and d' later in this proof. We know that the existence of a way which d-connects u and v given Z implies the existence of a path with the same property. It follows that $(u \not\perp\!\!\!\perp v \mid Z)_D$ holds.

Thus, our goal is to find a way w_{u-b} d-connecting u and b given Z in D which ends with $\to b$ under the assumption that there is a path $p'_{u-b} : u = v_1, v_2, \dots, v_{k-1} = a, v_k = b$ which d-connects u and b given Z in D' and ends with the edge $a \to b$. Let $d = v_i$ be the node with the minimal i such that $b \in \mathrm{De}(v_i)$ holds. Then either $d = u$ or we have $v_{i-1} \leftarrow d$ on p'_{u-b}. We will use this fact below to argue that there can be no collider at node d. Moreover, $v_j \notin Z$ holds for $i \leq j < k$, because none of these nodes is a collider on p'_{u-b} as we assumed that p'_{u-b} is a valid path d-connecting u and b given Z. In particular, $a \notin Z$ follows as $i \leq k-1$ holds for d. We will show that there is a path q_{d-b} d-connecting d and b which ends with $\to b$. Concatenating the subpath p'_{u-d} with this path q_{d-b} will result in the required way w_{u-b} because there can be no collider at node d and $d \notin Z$ holds.

The path q_{d-b} exists because of the following argument: The node d cannot be a descendant of b because then with a being a descendant of d, it would follow that a is a descendant of b and the edge $a \to b$ would not have been added to D. Moreover every node v_j with $j \geq i$ is not in Z as seen above. Then the statement $(d \not\perp\!\!\!\perp b \mid Z')_D$ holds for every subset Z' of Z because $(d \perp\!\!\!\perp b \mid Z')_D$ would imply the following contradiction: We know that $(a \not\perp\!\!\!\perp b \mid Z')_D$ (this holds for every Z' with $|Z'| \leq k$ because the edge $a \to b$ is in G) and also $(d \not\perp\!\!\!\perp a \mid Z')_D$ (because of the validity of the path p'_{u-a} and the fact that no node v_j with $j \geq i$ is in Z meaning the same follows for Z' as it is a subset). Note that because of the faithfulness of D, the same statements hold according to $\mathcal{I}_V^k$ as well. With $a \notin Z$ the edge $a \to b$ would have been removed from G_{ep} because these are exactly the conditions checked in line 3 of Algorithm 5. However, this would mean that we are not able to add the edge $a \to b$ to D'. A contradiction. This means that $(d \not\perp\!\!\!\perp b \mid Z')_D$ holds for every subset Z' of Z and therefore in particular for $Z' = Z \backslash \mathrm{De}(b)$. With Lemma 6.2 it follows that there is a path q_{d-b} d-connecting d and b given Z ending with $\to b$. □

The following Lemma is necessary to show that the edge maximal faithful DAGs have the same skeleton as G (Theorem 6.3 below).

Lemma 6.3. *If $a \leftarrow b \in G_{ep}$ and $a \rightarrow b \notin G_{ep}$, it follows that $b \notin De_D(a)$ holds for every DAG $D \in \mathcal{F}(\mathcal{I}_V^k)$.*

Proof. Having the edge $a \leftarrow b$ in G_{ep} but not the edge $a \rightarrow b$ implies that the following holds:

$$\exists Z \subseteq V \text{ with } |Z| \leq k \quad \exists u \in V \quad [(u \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V^k}, (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{I}_V^k}, (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V^k}, a \notin Z]$$

This is because these are exactly the conditions required to remove the edge $a \rightarrow b$ in line 4 of algorithm 5. From Theorem 4.2 we know that these conditions mean that no faithful DAG contains a causal path from a to b. Thus, $b \notin \text{De}_D(a)$ holds. □

We obtain one of the main results of this section that the edge maximal faithful DAGs have same skeleton as G. This means, as in the previous section, that, whenever two nodes a and b are nonadjacent in an edge maximal faithful DAG, there is either a statement $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V^k}$ with $|Z| \leq k$ or a and b are incompatible. This is due to the fact that these are the only two reasons an edge can be deleted in Algorithm 5.

Theorem 6.3. *The edge maximal DAGs faithful to $\mathcal{I}_V^k$ have the same skeleton as G_{ep}.*

Proof. This follows from Lemma 6.1, Theorem 6.2 and Lemma 6.3. The proof is identical to the one of Theorem 5.3 in the previous section. □

Of all DAGs faithful to $\mathcal{I}_V^k$ the edge maximal DAGs possess a very unique property. We will prove that these DAGs form a Markov equivalence class. This result has far reaching consequences. In order to show this we state that the edge maximal faithful DAGs have the same set of v-structures as G:

Theorem 6.4. *There is a v-structure in an edge maximal DAG $D \in \mathcal{F}(\mathcal{I}_V^k)$ iff it is in G_{ep}.*

Proof. We begin by showing that a v-structure in D, will also be in G_{ep}. Consider any v-structure $a \rightarrow c \leftarrow b$ in D. We note that, because a and c as well as c and b are adjacent in the faithful DAG D, the following holds:

$$\forall Z \text{ with } |Z| \leq k \quad (a \not\perp\!\!\!\perp c \mid Z)_{\mathcal{I}_V^k} \text{ and } (c \not\perp\!\!\!\perp b \mid Z)_{\mathcal{I}_V^k}$$

The nodes a and b are not adjacent in D and as D is edge maximal, it follows from the fact that G_{ep} and D have the same skeleton (Theorem 6.3) that they will not be adjacent in G_{ep} either. We will now show that the edges $a \leftarrow c$ and $c \rightarrow b$ are not in G_{ep}. Then we can conclude from the fact that every faithful DAG is contained in G_{ep} (Lemma 6.1) that the v-structure $a \rightarrow c \leftarrow b$ is in G_{ep}.

If the nodes a and b are not adjacent in G_{ep} there are two possible reasons for this:

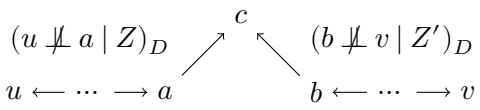

Figure 10: Case 2 of the proof of Theorem 6.4. The v-structure $a \to c \leftarrow b$ is in D and there exist Z and Z' such that $(u \not\perp\!\!\!\perp a \mid Z)_D$ and $(b \not\perp\!\!\!\perp v \mid Z')_D$ hold. We argue that $(u \not\perp\!\!\!\perp c \mid Z)_D$, $(v \not\perp\!\!\!\perp c \mid Z')_D$, $c \notin Z$ and $c \notin Z'$ hold as well.

1. The edge $a - b$ was not added to G_{ep} in line 1 because of an independence $(a \perp\!\!\!\perp b | Z)_{\mathcal{J}^k_V}$. Moreover, because we have $a \to c \leftarrow b$ in the faithful DAG D, it follows that $c \notin Z$ has to hold. This means that the edges are directed $a \to c \leftarrow b$ in G_{ep} because the conditions in line 3 of Algorithm 5 are met

   $$(a \perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V},\ (a \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}^k_V},\ (c \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V},\ c \notin Z$$

   and therefore the edges $a \leftarrow c$ and $c \to b$ were removed from G_{ep}.

2. The edges $a \to b$ and $a \leftarrow b$ were removed in line 4. This case is displayed in Figure 10.

   This means we have nodes u and v and sets Z and Z' such that

   $$\begin{aligned}&((u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V} \wedge (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{J}^k_V} \wedge (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V} \wedge a \notin Z)\\ \wedge\ &((v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}^k_V} \wedge (v \not\perp\!\!\!\perp b \mid Z')_{\mathcal{J}^k_V} \wedge (b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{J}^k_V} \wedge b \notin Z')\end{aligned}$$

   hold. Then $(u \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}^k_V}$ and $(v \not\perp\!\!\!\perp c \mid Z')_{\mathcal{J}^k_V}$ hold as well because of $a \notin Z$ and $b \notin Z'$ and the fact that with the edges $a \to c$ and $b \to c$ in the faithful DAG D there is neither a collider at node a nor at node b. On the other hand there is a collider at node c (one the path from u to b as well as from v to a) and therefore $c \notin Z$ and $c \notin Z'$ hold. Then with $(c \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}$ (with the edge $c \leftarrow b$ in D there cannot be any independence) and $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}^k_V}$ the edge $c \to b$ is removed from G because the conditions in line 3 of Algorithm 5 are met. The edge $a \leftarrow c$ is removed, too, as additional to $(v \not\perp\!\!\!\perp c \mid Z')_{\mathcal{J}^k_V}$ and $c \notin Z'$ the statements $(a \not\perp\!\!\!\perp c \mid Z')_{\mathcal{J}_V}$ and $(v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V}$ hold. Therefore we have the v-structure $a \to c \leftarrow b$ in G as well.

Now we show that a v-structure in G_{ep} will be present in D as well. It follows from Lemma 6.1 that if we have $a \to c \leftarrow b$ in G_{ep}, D can neither contain an edge between a and b nor the edges $a \leftarrow c$ or $c \to b$. Moreover because of the edge maximality of D and the fact that D and G_{ep} have the same skeleton (Theorem 6.3) that the edges $a \to c \leftarrow b$ will be present in D. □

We will now include the Meek rules in our argument in order to show the following important result which shows a way to obtain faithful DAGs from the graph G which is the final result of Algorithm 5.

Corollary 6.1. *The set of edge maximal DAGs faithful to $\mathcal{I}_V^k$ is the Markov equivalence class formed by all consistent extensions of the CPDAG G.*

Proof. Theorem 6.3 states that the edge maximal DAGs have the same skeleton and Theorem 6.4 states that they have the same set of v-structures as G_{ep}. Thus, these DAGs form a Markov equivalence class which is exactly the set of all consistent extensions of G_{ep}. It immediately follows that the graph G which results from applying the Meek rules to G_{ep} is a CPDAG. Moreover, by correctness of the Meek rules (these rules neither create a new v-structure nor a cycle [12]) G has the same set of consistent extensions as G_{ep}. □

Finally, it becomes clear why we can apply the three rules R1-R3 to G_{ep} in Algorithm 5. As shown by Meek [12] these rules maximally extend a PDAG whose consistent extensions form a Markov equivalence class into a CPDAG and the edge maximal DAGs are the Markov equivalence class formed by the consistent extensions of G_{ep}. If an edge $a \to b$ gets directed by the rules R1-R3, this means that it is in every consistent extension (while the edge $a \leftarrow b$ is in no consistent extension). That the application of these three rules is correct for all faithful DAGs — not only the edge maximal ones — will be argued in the proof of the following theorem. This theorem combines the previous results to show that G represents the set of faithful DAGs.

Theorem 6.5. *The CPDAG G is the representation of the set $\mathcal{F}(\mathcal{I}_V^k)$.*

Proof. A representation G of the set $\mathcal{F}(\mathcal{I}_V^k)$ is a minimal graph that contains every graph in $\mathcal{F}(\mathcal{I}_V^k)$. We begin by proving that G indeed contains every DAG faithful to $\mathcal{I}_V^k$. We do this by showing that every faithful DAG is a subgraph of a consistent extension of G. Consider the faithful DAG $D \in \mathcal{F}(\mathcal{I}_V^k)$. The DAG D has to be a subgraph of some edge maximal faithful DAG. We know from Corollary 6.1 that every edge maximal DAG is a consistent extension of G . Thus, D is a subgraph of a consistent extension of G.

We show now that G is indeed minimal. This holds as deleting or directing an edge in G would immediately violate the condition that G contains every faithful DAG. This follows as we know that there are (edge maximal) faithful DAGs with the same skeleton as G and the fact that G is a CPDAG representing a Markov equivalence class. □

From the above theorems we can deduce an interesting fact. Theorem 6.2 holds for G_{ep} and not only for G and the only constraint we impose on adding edges is that they do not produce a cycle. Thus, if we have an edge $a - b$ in G_{ep}, an edge $a \to b$ in G (meaning the edge $a \to b$ has been directed by one of the Meek rules (R1-R3)) and a DAG $D \in \mathcal{F}(\mathcal{I}_V^k)$, it follows that either $a \to b$ is in D or there is a causal path from a to b. Moreover we derive the following remarkable result:

Corollary 6.2. *The representation G of a set $\mathcal{F}(\mathcal{I}_V^k)$ is a CPDAG.*

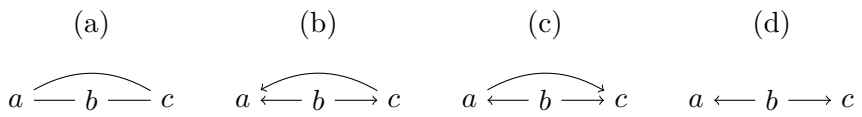

Figure 11: The representation G is shown in (a). Two different consistent extensions of G are displayed in (b) and (c). After removing the edge between a and c the resulting faithful DAG (d) is the same.

This follows immediately from Corollary 6.1. It implies that the notion of a representation we introduced in this thesis is a generalization of the notion of a CPDAG. More precisely, for every k there is a subclass of CPDAGs (let us call these k-CPDAGs) which are the representation of a set of DAGs faithful to a set $\mathcal{I}_V^k$ for a fixed $|V| = n$. In particular, the set of l-CPDAGs is a subset of the set of $l+1$-CPDAGs. Further investigations of these structures might be interesting, for example, for the open question of counting the number of Markov equivalence classes (which is equal to the number of CPDAGs) for a given number n of nodes [15]. Notably, Textor et al. [18] analyzed the number of 0-CPDAGs (they use a different representation termed SMIG).

6.4 Enumerating all faithful DAGs

After obtaining the representation G of the set of faithful DAGs $\mathcal{F}(\mathcal{I}_V^k)$ the question arises how these faithful DAGs can be found and, more precisely, how they can be enumerated. For the case of marginal independences this problem has already been studied by Textor et al. [18]. For the general case, on the other hand, this problem is new.

The results in the previous subsection give insights into the structure of the set of faithful DAGs. Of central importance is the result that the consistent extensions of the representation G are the edge maximal DAGs faithful to $\mathcal{I}_V^k$. This shows that the notion of a representation is useful for finding the faithful DAGs. It allows us to do this with the following approach. We start with every consistent extension of G and output it. Then we recursively delete an edge and check if the produced DAG is still faithful. If it is, we output the DAG and continue the recursion. Else we stop the recursion. With this approach we find all faithful DAGs, because we know that these are subgraphs of a consistent extension of G and that all *intermediate* DAGs are faithful as well. With intermediate graphs we mean the following. Consider that D_1 is a consistent extension of G and D_2 is a subgraph of D_1 which is also faithful to $\mathcal{I}_V^k$. Clearly, $E_2 \subseteq E_1$ holds. Then we know from Theorem 6.2 that every DAG D_3 lying between these two DAGs with $E_2 \subseteq E_3 \subseteq E_1$ will be faithful as well. This is because all the edges in $E_1 \backslash E_2$ are in G and adding them will not produce a cycle. We can conclude that it is possible starting with consistent extensions of G to obtain every faithful DAG by stepwise edge removal.

The problem of this approach is that we would output certain DAGs multiple times as

Figure 11 shows. Given the representation G of the set $\mathcal{I}_V^0 = \emptyset$ shown in (a), we present two consistent extensions in (b) and (c). After removing the edge between a and c the result is the same faithful DAG in both cases which is shown in (d). Therefore, it remains an open problem if there is an algorithm which enumerates all faithful DAGs similar to the one given by Textor et al. [18] we mentioned in the previous section. Their algorithm makes extensive use of the edge minimal faithful DAGs. In this thesis we deal with the edge maximal faithful DAGs intensively. The edge minimal DAGs, on the other hand, are much harder to classify in the general setting of CI up to order k. It would be very interesting to study this special class of DAGs in more detail which might help in solving the stated enumeration problem.

7 The OPPC algorithm

In the previous section the goal was to find faithful DAGs for a set of CIs with a bounded order. We will now consider the case that we have complete conditional independence information through a set $\mathcal{I}_V$. This set can also be viewed as an oracle telling us if a certain conditional independence is present. As already discussed in Section 3.3, there are a number of algorithms tackling this problem. The focus lies mainly on minimizing the number of CI tests the algorithm carries out. Especially high order CI tests are undesirable due to their weak statistical properties. With these goals in mind, we introduce a modified version of the PC algorithm we call the OPPC (One-Phase Pattern Construction) algorithm.

7.1 Idea of the OPPC algorithm

We begin by recalling the different phases (as stated by Pearl [13]) of classical constraint-based causal structure learning algorithms. In particular, the PC algorithm is one of these algorithms and consists of exactly the steps stated below (for more details on the algorithm see Section 3.3).

1. Learning the skeleton G_{skel}.
2. Orientation of v-structures (obtaining a pattern G_{patt}).
3. Orientation of remaining edges with the Meek rules (producing the final result G).

First, the PC algorithm learns the skeleton. This is done by searching for independences starting from a complete graph and removing edges accordingly. It holds for the skeleton of the true DAG that there is a CI in $\mathcal{I}_V$ for every pair of nonadjacent nodes a and b. In particular, there is such a CI which is solely formed by neighbors of a or by neighbors of b. The PC algorithm which exhaustively goes through all possible separating sets Z formed by the neighbors will therefore find this CI, store the set Z and remove the corresponding edge. Thus, the PC algorithm learns the correct skeleton. Second, the algorithm orients the v-structures using the stored separating sets Z. The result of this step is called the pattern of the underlying DAG. All v-structures are directed and every directed edge is part of a v-structure. Third, the remaining edges are oriented with the Meek rules producing the final result G. The separation of the learning of the skeleton and the orientation of the edges is the key paradigm for all classical constraint-based causal structure learning algorithms. We will propose a completely new approach which only consists of two steps:

1. Learning of the extended pattern (obtaining G_{ep}).
2. Orientation of the remaining edges with the Meek rules (producing the final result G).

In particular, the new idea is that we merge the learning of the skeleton and the orientation of the v-structures. We call the result of this first step the *extended pattern*. This is not purely a pattern as some additional edges can be directed which are not part of a v-structure. In the example shown in Figure 12 we will investigate this phenomenon further. The important fact for now is, however, that all necessary v-structures are indeed oriented. In the second step the remaining edges are oriented by application of the Meek rules. We introduce the main ideas that enable us to combine the learning of the skeleton and the orientation of the v-structures:

1. We are able to direct edges during the search for (conditional) independences.

   In the PC algorithm we start by learning the skeleton and direct edges afterwards. There, directing steps are done in the following manner. For every chain $a - c - b$ in the skeleton we check if c is in the set $S(a, b)$ which separated a and b. If c is not in $S(a, b)$ we direct the edges to form the v-structure $a \to c \leftarrow b$. Note that in this case we have to store the set $S(a, b)$ when discovering the independence $(a \perp\!\!\!\perp b \,|\, S(a, b))_{\mathcal{J}_V}$, in order to use this set later on.

   However, it is also possible to direct the edges immediately when finding a set Z which separates a and b. We have seen this idea throughout the thesis. If we want to direct the edges $a - c$ and $c - b$ given an independence $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ we need to check that for the node c the following conditions are satisfied:

   $$(a \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}_V},\ (c \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V} \text{ and } c \notin Z$$

   These conditions were previously stated in Theorem 4.1. If they hold, the edges can only appear as $a \to c$ and $c \leftarrow b$ in any faithful DAG. Clearly, the cost of finding these directions are the two additional CI tests we have to conduct in order to check that $(a \not\perp\!\!\!\perp c \,|\, Z)_{\mathcal{J}_V}$ and $(c \not\perp\!\!\!\perp b \,|\, Z)_{\mathcal{J}_V}$ hold. These were not needed in the PC algorithm. There we know implicitly that these conditions hold because an edge $a - c$ (or $c - b$) in the skeleton indicates that these nodes cannot be separated by any set and thus, in particular not by Z. However, as we start directing edges before the skeleton is known in the OPPC algorithm, we have to explicitly test if these statements hold.

2. We can delete edges between incompatible nodes.

   If an edge can neither be directed $a \to b$ nor $a \leftarrow b$ because in either case the conditions from Theorem 4.1 mentioned above are fulfilled, the nodes a and b are nonadjacent in the true skeleton. This is because the nodes a and b are incompatible. Thus, we are able to delete the edges $a \to b$ and $a \leftarrow b$ — thereby making a and b nonadjacent — without having found a separating set Z for which $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ holds. This means in turn that we can remove edges earlier than in the PC algorithm. However, it is vital for the correctness of this approach that all the v-structures $a \to c \leftarrow b$ can be directed even without the set Z which separates a and b. We have

already seen that a similar statement (see Theorem 6.4) holds for the case of a set of independences bounded by the order k. That this is the case here as well is the most important observation and is proven below in Theorem 7.2.

We can immediately see why combining the learning of the skeleton and the orientation of the v-structures is able to improve the efficiency of causal structure learning. Directing the edges earlier will mean that we are also able to delete edges earlier than in the PC algorithm. This means that the search space for forming separating sets (these sets are formed by the neighboring nodes) is smaller. Therefore, the OPPC algorithm learns the true CPDAG more efficiently than the PC algorithm.

We have already seen similar ideas in the previous section and we can easily connect the OPPC algorithm with Algorithm 5. Clearly, in order to solve the problem of recovering causal structures from a complete set of conditional independence information $\mathcal{J}_V$ one could also use Algorithm 5 (imagine that we set $k = n - 2$). However, this algorithm goes through the whole set $\mathcal{J}_V$ exhaustively. The OPPC algorithm does this (similarly to the PC algorithm) in an iterative fashion forming separating sets Z only from adjacent nodes which makes it much more efficient. In particular, one could also view the OPPC algorithm as a special version Algorithm 5 which considers the CIs in a certain iterative order. Moreover, we see the connection of the notion of a representation to causal structure learning. The representation was the minimal graph containing all faithful DAGs. A constraint-based causal structure learning algorithm is the most efficient if the graph which is currently considered — meaning which contains all DAGs faithful to the observed CIs — has as few edges as possible because this reduces the search space. This is exactly what we required from the representation. Of course, as mentioned above, the difference is that in the previous section, we actually considered all CIs as long as the order was less or equal to k while here we have the additional constraint that the separating sets Z are only formed by the neighbors.

7.2 The OPPC algorithm

The OPPC algorithm is presented as Algorithm 6. We have already seen the ideas behind this algorithm. Now we go through the details of the implementation and consider a detailed example afterwards. Starting with a complete graph the algorithm looks for independences which will enable us to remove edges from the graph. This is done by iteratively increasing the order l of the considered CIs beginning with order zero. In each iteration we consider all currently adjacent nodes i and j and try to separate them. We do this by exhaustively going through all separating sets Z of cardinality l (the currently considered order) which can be formed by the neighbors of i. If we find that $(i \perp\!\!\!\perp j \mid Z)_{\mathcal{J}_V}$ holds, we remove the edges $i \to j$ and $i \leftarrow j$. This is done just as in the PC algorithm. The main difference comes afterwards. We immediately try to orient further edges by checking if the conditions stated in Theorem 4.1 apply. We note that orienting an edge $a - b$ into $a \to b$ actually means removing the edge $a \leftarrow b$. We use the term orientation at times because it

input : Vertex Set V, Conditional Independence Information $\mathcal{I}_V$
output: CPDAG G representing the Markov equivalence class of faithful DAGs

Let G_{ep} be the complete undirected graph on the vertex set V.
Set $l = -1$.
repeat
  Set $l = l + 1$.
  repeat
    Select a new ordered pair of nodes i, j with i and j adjacent such that $|N(i)\backslash\{j\}| \geq l$.
    repeat
      Choose new $Z \subseteq N(i)\backslash\{j\}$ with $|Z| = l$.
      if $(i \perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V}$ **then**
        Remove $i \to j$ and $i \leftarrow j$ from G_{ep}.
        foreach $k \in (N(i) \cup N(j))\backslash(\{i, j\} \cup Z)$ **do**
          if $(i \not\perp\!\!\!\perp k \mid Z)_{\mathcal{I}_V}$ *and* $(k \not\perp\!\!\!\perp j \mid Z)_{\mathcal{I}_V}$ **then**
            Remove $i \leftarrow k$ and $k \to j$ from G_{ep}.
          end
        end
      end
    until *i and j are not adjacent or all $Z \subseteq N(i)\backslash\{j\}$ with $|Z| = l$ have been chosen*
  until *all ordered pairs of adjacent variables i and j such that $|N(i)\backslash\{j\}| \geq l$ have been selected*
until *for each ordered pair i, j with i and j adjacent: $|N(i)\backslash\{j\}| < l$*
Set $G = G_{\text{ep}}$.
repeat
  R1: In G orient $j - k$ into $j \to k$ whenever there is an arrow $i \to j$ such that i and k are nonadjacent.
  R2: In G orient $i - j$ into $i \to j$ whenever there is a chain $i \to k \to j$.
  R3: In G orient $i - j$ into $i \to j$ whenever there are two chains $i - k \to j$ and $i - l \to j$ such that k and l are nonadjacent.
until *No further edges can be oriented in G by application of R1-R3.*

Algorithm 6: The OPPC algorithm

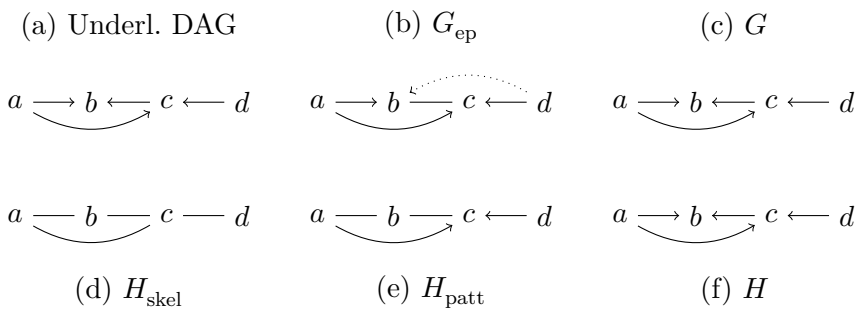

Figure 12: A comparison of the PC and the OPPC algorithm for learning the DAG shown in part (a). The steps of the OPPC algorithm are shown in (b) and (c) in the upper row (G_{ep} and G). The steps of the PC algorithm are shown in (d), (e) and (f) in the lower row (H_{skel}, H_{patt} and H). In particular, we emphasize the difference between G_{ep} and H_{patt}.

is more intuitive, but formally edges are removed. We do this by going through all nodes k which are not in Z and a neighbor of i or j (else we would not be able to remove an edge). This step is similar to the orientation of the v-structures in the PC algorithm, but there are some major differences due to the fact that we do this before knowing the true skeleton. In particular, the interesting thing to consider is what happens if a and b become nonadjacent not because a separating set is found, but because the edges $a \to b$ and $a \leftarrow b$ are removed in different iterations of line 13. If this happens, the nodes a and b are incompatible. We will show later in this section that even in this case all v-structures $a \to c \leftarrow b$ are detected.

At some point no further separating sets can be formed because l is larger or equal to the number of neighbors of i. In this case we have successfully learned the extended pattern (we go into more detail why this pattern is extended in the following subsection) and maximally extend the graph with the Meek rules. Note that we store the extended pattern in G_{ep}. We explicitly need this graph for some proofs later in this section.

7.3 Example and comparison with the PC algorithm

In this subsection we use the example in Figure 12 to illustrate an execution of the OPPC algorithm and compare it with the PC algorithm. Particular emphasis lies on comparing the pattern occurring in the PC algorithm with the extended pattern in the OPPC algorithm. The underlying DAG is shown in part (a). The corresponding set of independences is the following:

$$\mathcal{I}_V = \{(a \perp\!\!\!\perp d), (b \perp\!\!\!\perp d \mid \{a, c\})\}.$$

We will denote the graphs used during the execution of the PC algorithm with H and the graphs used during the execution of the OPPC algorithm with G. We start by investigating how the PC algorithm learns the causal structure (this is shown in the lower row of

Figure 12). The PC algorithm begins by learning the skeleton of the underlying DAG. The algorithm starts with a complete graph and removes edges when independences are found. Here, we find the independences $(a \perp\!\!\!\perp d)_{\mathcal{J}_V}$ (for $l = 0$) and $(b \perp\!\!\!\perp d \mid \{a, c\})_{\mathcal{J}_V}$ (for $l = 2$) and therefore remove the edges $a - d$ and $b - d$. We obtain the intermediate result H_{skel} shown in (d). In particular, we note that the set $\{a, c\}$ is formed of the neighbors of b. The separating sets $S(a, d) = \emptyset = S(d, a)$ and $S(b, d) = \{a, c\} = S(d, b)$ are stored. Afterwards, the algorithm considers every chain $i - k - j$ in H_{skel}. In this case, we have the chains $a - c - d$ and $b - c - d$. The chain $a - c - d$ gets oriented as a v-structure because c is not in $S(a, d)$. The chain $b - c - d$ on the other hand is not oriented because c is in $S(b, d)$. We obtain the next intermediate result H_{patt} shown in (e). Finally, the Meek rules (R1-R3) are applied until no further edges can be oriented. Due to the first Meek rule the edge $b - c$ is oriented as $b \leftarrow c$ and due to the second Meek rule the edge $a - b$ is oriented as $a \rightarrow b$. The result is a CPDAG H (shown in (f)) which in this case coincides with the underlying DAG.

In the OPPC algorithm the first two phases of the PC algorithm are combined. Immediately after finding the independence $(a \perp\!\!\!\perp d)_{\mathcal{J}_V}$ and removing the edges $a \rightarrow d$ and $a \leftarrow d$, the algorithm checks if further edges can be directed. To do this the algorithm goes through every possible third node. More precisely, we look at every such node k which could actually lead to the orientation of an edge. This is the case if $k \in (N(i) \cup N(j)) \backslash (\{i, j\} \cup Z)$ holds. We note that orienting an edge $u - v$ into $u \rightarrow v$ in the considered graph model is equivalent to removing the edge $u \leftarrow v$. The two nodes which are considered as k are b and c. Thus, it is tested if $(a \not\perp\!\!\!\perp c)_{\mathcal{J}_V}$, $(c \not\perp\!\!\!\perp d)_{\mathcal{J}_V}$ and $c \notin \emptyset$ hold. This is the case and therefore the edges $a \leftarrow c$ and $c \rightarrow d$ are removed. In other words, we orient the edges as $a \rightarrow c$ and $c \rightarrow d$. The same test is done for b. Here as well $(a \not\perp\!\!\!\perp b)_{\mathcal{J}_V}$, $(b \not\perp\!\!\!\perp d)_{\mathcal{J}_V}$ and $b \notin \emptyset$ hold. The edges $a \leftarrow b$ and $b \rightarrow d$ are removed or, in other words, the edges get directed as $a \rightarrow b$ and $b \leftarrow d$. Afterwards, we find the conditional independence $(b \perp\!\!\!\perp d \mid \{a, c\})_{\mathcal{J}_V}$. Consequently, the edges $b \leftarrow d$ and $b \rightarrow d$ are removed (note that the edge $b \rightarrow d$ has already been removed previously). Again we check if further edges can be oriented. But here a and c are both in the separating set $\{a, c\}$ and therefore not considered as k. When looking at the result G_{ep} we see that it is not identical to the pattern H_{patt} obtained by the PC algorithm. The edge $a \rightarrow b$ is directed even though it is not in a v-structure. This is why we introduced the term extended pattern. After following this example it is also clear why edges like $a \rightarrow b$ can be oriented without being in a v-structure. The reason is that this edge is part of a disguised v-structure $a \rightarrow b \leftarrow d$ (this is indicated by the dotted arrow in G_{ep} above). However, the edge $b \leftarrow d$ was removed during the execution of the algorithm. This shows that due to the new ideas in the OPPC we are able to direct more edges when constructing the pattern compared to the PC algorithm. We note that the edge $a - b$ is directed $a \rightarrow b$ in the PC algorithm as well. This, however, happens not until the application of the Meek rules. These Meek rules are applied in the OPPC algorithm as well to orient the remaining edges. Just as in the PC algorithm, the edge $b - c$ is oriented as $b \leftarrow c$ due to the first Meek rule.

We will now analyze the number of CI tests needed in the OPPC algorithm. We will measure this in the cardinality of the vertex set n dependent on the maximum degree in the underlying DAG. We assume that we have oracle access to the complete conditional independence information. The same worst case as for the PC algorithm applies. However, in line 12 the OPPC algorithm employs two additional independence tests for each neighbor of i or j whenever an edge $i - j$ is deleted. As it is possible that $\mathcal{O}(n^2)$ edges are deleted and there are at most $\mathcal{O}(n)$ neighbors, an additional additive term $\mathcal{O}(n^3)$ follows. Asymptotically the number of CI tests is therefore bounded by $\mathcal{O}(n^{2+d_{\max}} + n^3)$ (see the analysis in Section 3.3). We note here that through the tests in line 12 it is possible that tests are executed redundantly, meaning we test a statement which has already been tested in line 9 (or the other way around). We will discuss this issue further in the subsequent section. While the OPPC algorithm does not improve the worst case number of necessary CI tests, it is clear that for most cases incompatible nodes are found and thus, edges are deleted earlier in comparison to the PC algorithm leading to a reduced search space. In the subsequent section we will experimentally show that this allows for more efficient causal structure learning in the average case.

7.4 Proof of correctness

In this subsection we formally prove the correctness of the OPPC algorithm. We begin by proving the following statements for the PDAG which is obtained after line 19 of the OPPC algorithm (we call this PDAG G_{ep}). We show that this PDAG has the same skeleton and the same v-structures as every faithful DAG. Afterwards, we will include the rules R1-R3 in our argument which extend G_{ep} maximally into a CPDAG. Showing that the OPPC algorithm is correct, is much more straightforward than the proofs in the previous sections. It is easy to show that we find the correct skeleton which moreover is the same for all faithful DAGs. This is due to the fact that all edge removals are clearly correct and that we have the comfort of having access to all CIs. This means that we can be sure to find an independence for every edge which is not in the true skeleton. That all v-structures are oriented correctly, on the other hand, is the central theorem of this section. This holds for chains $a - c - b$ if the edges $a \to b$ and $a \leftarrow b$ were removed because of a CI $(a \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ (see line 13). Showing that this also holds if a and b are incompatible nodes is more intricate, but similar to the proof of Theorem 6.4.

Theorem 7.1. *The PDAG G_{ep} has the same skeleton as every faithful DAG $D \in \mathcal{F}(\mathcal{J}_V)$.*

Proof. The above statement can also be formulated as: The nodes a and b are nonadjacent in PDAG G_{ep} *iff* they are nonadjacent in every faithful DAG D. First, we show that if an edge is missing from PDAG G_{ep}, it is also missing from D. Essentially this means that whenever during the execution of the algorithm an edge was removed from G_{ep}, this choice was correct. There are two cases to consider:

1. The edge was removed in line 10 of the algorithm. This happens only if a set Z was found that separates a and b. But then the nodes a and b cannot be adjacent in any faithful DAG as else a and b would be conditionally dependent given Z.

2. The edges $a \to b$ and $a \leftarrow b$ were removed in line 13. Then the following holds:

$$(\exists u, Z)\ ((u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V} \wedge (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{J}_V} \wedge (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V} \wedge a \notin Z)$$
$$\wedge\ (\exists v, Z')\ ((v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V} \wedge (v \not\perp\!\!\!\perp b \mid Z')_{\mathcal{J}_V} \wedge (b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V} \wedge b \notin Z')$$

   Thus, the nodes a and b are incompatible. It follows from Corollary 4.1 that there cannot be an edge between a and b in any faithful DAG.

Second, we show that if a and b are nonadjacent in a faithful DAG D, they are also nonadjacent in PDAG G_{ep}.

We assume for the sake of contradiction that there is an edge between a and b in PDAG G_{ep}. Then there was no set Z found in line 8 which separated a and b. But we exhaustively searched for every subset of the set of neighbors of a (and b) in G_{ep}. This set of neighbors was at all times a superset of the neighbors of a (and b) in D because we have seen in the first part of this proof that if a and b are nonadjacent in G_{ep} they are also in D. But this means that no subset of neighbors of a (or b) separates a and b in D which would mean that a and b are adjacent in D as well. A contradiction. □

The following theorem is central for the proof of the whole algorithm. It shows that the v-structures can be successfully reconstructed even if no separating set was found.

Theorem 7.2. *There is a v-structure in the PDAG G_{ep} resulting from Algorithm 6* iff *there is a v-structure in a faithful DAG $D \in \mathcal{F}(\mathcal{J}_V)$.*

Proof. First, we show that if there is a v-structure in a faithful DAG D, then there will be a v-structure in the PDAG G_{ep}. Let the v-structure we consider be $a \to c \leftarrow b$. We have already seen in Theorem 7.1 that the edge $a - b$ will be deleted in PDAG G_{ep}. It remains to be shown that the edges are actually directed $a \to c$ and $c \leftarrow b$. We consider two cases:

1. The edge $a - b$ was deleted in line 10 because a separating set Z was found. We will show that the conditions in the if-statement in line 12 are met.

   Because there is the structure $a \to c \leftarrow b$ in the faithful DAG D, it follows that $(a \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}_V}$ and $(c \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ hold and c is not in Z (note that this holds for every separating set Z). Thus, the edges $a \leftarrow c$ and $c \to b$ are removed in line 13. Due to the fact that G_{ep} and D have the same skeleton (Theorem 7.1) the edges $a \to c \leftarrow b$ will not be removed later.

2. The edges $a \leftarrow b$ and $a \to b$ were deleted (at different iterations) in line 13 of the algorithm. In other words, the nodes a and b are incompatible. Thus, the following

holds for two sets Z and Z' because of the conditions in line 12::

$$((u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V} \wedge (u \not\perp\!\!\!\perp a \mid Z)_{\mathcal{J}_V} \wedge (a \not\perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V} \wedge a \notin Z)$$
$$\wedge ((v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V} \wedge (v \not\perp\!\!\!\perp b \mid Z')_{\mathcal{J}_V} \wedge (b \not\perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V} \wedge b \notin Z')$$

Moreover, we have a v-structure $a \to c \leftarrow b$ in D. We know from case 2 in the proof of Theorem 6.4 that $(u \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}_V}$, $(v \not\perp\!\!\!\perp c \mid Z')_{\mathcal{J}_V}$, $c \notin Z$ and $c \notin Z'$ hold. But then the edges $a \leftarrow c$ and $c \to b$ are removed in line 13 when the independences $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ and $(v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V}$ were found. This is because when finding $(u \perp\!\!\!\perp b \mid Z)_{\mathcal{J}_V}$ we looked at every node in $(N(u) \cup N(b)) \backslash (\{u, b\} \cup Z)$ and thus, in particular, we looked at node c as it is a neighbor of b and not in Z. For this node c the conditions in line 12 are fulfilled. Analogously, when finding $(v \perp\!\!\!\perp a \mid Z')_{\mathcal{J}_V}$ we looked at every node in $(N(v) \cup N(a)) \backslash (\{v, a\} \cup Z'))$ and therefore, in particular, at node c as it is a neighbor of a and not in Z'.

Second, we show that if there is a v-structure in the PDAG G_{ep}, this v-structure is also in every faithful DAG. Let the v-structure we consider be $a \to c \leftarrow b$. We know from Theorem 7.1 that the edge $a - b$ will be missing in every faithful DAG and an edge between a and c as well as c and b is present in every faithful DAG. We know that when only the edge $a \to c$ (but not $a \leftarrow c$) is in G_{ep} the following holds for some u and Z:

$$(u \perp\!\!\!\perp a \mid Z)_{\mathcal{J}_V},\ (u \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}_V},\ (a \not\perp\!\!\!\perp c \mid Z)_{\mathcal{J}_V} \text{ and } c \notin Z.$$

This means that Theorem 4.1 applies and no faithful DAG can contain the edge $a \leftarrow c$ and accordingly has to contain the edge $a \to c$. The same holds for $c \leftarrow b$. Thus, in every faithful DAG there will also be the v-structure $a \to c \leftarrow b$. □

We are now able to conclude that Algorithm 6 returns the correct CPDAG.

Theorem 7.3. *Algorithm 6 produces the CPDAG consistent with the given conditional independence information $\mathcal{F}(\mathcal{J}_V)$.*

Proof. We have shown in Theorem 7.1 that the algorithm produces the same skeleton as every faithful DAG. Moreover the PDAG G_{ep} we obtain after line 19 contains the same v-structures as every faithful DAG and every directed edge $a \to b$ is directed in this manner in every faithful DAG. Then the three rules in lines 22- 24 maximally extend this PDAG as shown by Meek [12] to produce the correct CPDAG G. □

8 Experimental analysis of the OPPC algorithm

The goal of the OPPC algorithm was, on one hand, to reduce the number of needed CI tests and, on the other hand, to mainly use low-order CIs which are statistically easier to test than CIs of higher order. In this section, we test experimentally if we reached this goal.

8.1 Experimental setting

For our tests we generate a random DAG D which represents the causal structure we want to learn and which substitutes the set $\mathcal{I}_V$ as this would be far too large to store. Whenever we have to test a CI statement $(i \perp\!\!\!\perp j \,|\, Z)$, we will test if i is d-separated from j by Z in D.

We use different models to generate D. First, we generate random regular DAGs. For this a regular graph G with a certain degree d is randomly chosen and afterwards the edges are directed by imposing a random topological sorting on G. Second, we generate graphs with a fixed edge probability p — meaning a certain edge is present with probability p. This is also known as the Erdős–Rényi model. For edge directions we will again randomly choose a topological sorting. As implementation for these methods we use the R-package `pcalg` [10] which provides the method `randDAG` to generate random DAGs. Note that for generating regular graphs `randDAG` uses the package `igraph` [6]. For performance reasons we implemented the PC and the OPPC algorithm in C++. In order to test d-separation we used our own implementation of the Bayes-Ball algorithm [16]. The usage of R would have been too slow for meaningful tests considering that our C++ implementation is more than a factor 1000 faster than the standard implementation in the package `pcalg`. One example to illustrate this: For a DAG generated by the random regular graph model above with 26 nodes and node degree 5, the `pcalg` implementation of the PC algorithm runs for $6,882$ second which is almost 2 hours. Our C++ implementation on the other hand finishes after 4.1 seconds. These tests were run on one core of the Intel i5-2540M processor with a 2.6 GHz clock rate.

For the conducted tests we present the number of carried out CI tests instead of the execution time of the algorithms. The reason for this is that, as we test for d-separation instead of using statistical CI tests, this time would not be very meaningful for practical applications. Moreover, it is safe to assume that the number of CI tests is a good indicator for the running time in practice as these tests constitute the by far largest part of the computational effort.

8.2 Results in the regular graph model

In Table 1 we present the results of applying the PC algorithm and the OPPC algorithm to regular DAGs. The first thing to notice is that for regular graphs with a node degree smaller or equal to four the PC algorithm uses fewer CI tests than the OPPC algorithm.

Table 1: The number of CI tests carried out by the PC and the OPPC algorithm for regular DAGs. The shown values are the median of 50 random trials. OPPC (uq.) counts the number of unique tests a run of the OPPC algorithm executes. On the right we present the number of deleted edges due to incompatible nodes during a run of the OPPC algorithm contrasted with the total number of deleted edges. More precisely, if both $a \to b$ and $a \leftarrow b$ have been removed we count this as one deleted edge.

DAG		Number of CI tests			Deleted edges (OPPC)	
n	d	PC	OPPC	OPPC (uq.)	Edges betw. IN	All edges
50	3	4,786	69,650	3,668	138	1,150
100	3	11,606	597,957	9,719	336	4,800
150	3	21,022	2,076,385	18,786	547	10,950
200	3	35,682	4,999,257	30,897	803	19,600
250	3	48,208	9,883,800	44,188	1,018	30,750
300	3	65,554	17,196,809	59,179	1,272	44,400
25	4	34,620	19,260	12,641	63	250
50	4	122,061	74,223	22,233	308	1,125
75	4	277,330	226,112	45,318	677	2,625
100	4	377,898	512,954	50,204	1,123	4,750
125	4	417,558	1,006,377	59,851	1,576	7,500
150	4	445,912	1,742,788	65,854	2,050	10,875
26	5	534,286	279,953	-	69	260
50	5	5,682,999	1,092,363	-	356	1,100
76	5	21,471,567	2,500,688	-	901	2,660
100	5	56,936,382	5,648,846	-	1,600	4,700
126	5	91,237,366	5,901,799	-	2,505	7,560
150	5	153,242,808	9,783,326	-	3,540	10,800
25	6	2,074,363	1,012,268	-	52	225
50	6	105,862,939	29,351,574	-	344	1,075
75	6	875,703,600	180,450,254	-	919	2,550

We will explain the reason behind this and show how this might be resolved. For node degree greater than four, however, the OPPC algorithm performs better.

That the OPPC algorithm needs more CI tests for graphs with a small node degree d has the following reason. As analyzed above the OPPC algorithm has an additional additive term $\mathcal{O}(n^3)$. This means that whenever the number of CI tests used by the PC algorithm grows slower than $\mathcal{O}(n^3)$, the standard implementation of the OPPC algorithm cannot perform better. This can be seen very well for $d = 3$. When the number of nodes is doubled from 100 to 200 the number of CI tests carried out by the OPPC algorithm increases roughly eightfold (which fits very well with 8 being 2^3) indicating that the running time is dominated by the term $\mathcal{O}(n^3)$. The growth of the PC algorithm on the other hand is much slower and less than quadratic.

However, as mentioned in the previous section, the additional CI tests in the OPPC algorithm might be redundant. For this reason we also present the number of unique CI tests which were executed by the OPPC algorithm. This number is much smaller and indeed smaller than the number of CI tests used in the PC algorithm showing the potential of the OPPC algorithm. In this context we have to mention the fact that the PC algorithm, too, tests CIs redundantly. This can happen as ordered nodes are selected (see line 14 of Algorithm 2 in Section 3.3) and we might test for $(i \perp\!\!\!\perp j \mid Z)$ as well as $(j \perp\!\!\!\perp i \mid Z)$. It is clear, however, that these redundancies in the worst case lead to a doubling of the number of necessary CI tests. For the OPPC algorithm on the other hand, these redundant CI tests are much more of a nuisance and it is an important question how these can be reduced without simply storing the results of these tests which would lead to much increased memory demand.

On the other hand, the tests confirm that the redundant CI tests are indeed negligible for graphs with a larger node degree when the number of CI tests grows faster than $\mathcal{O}(n^3)$. This can be seen by the results for regular graphs with node degree 5. Here the OPPC algorithm performs clearly better than the PC algorithm. We can see that the number of needed CI tests grows much faster for the PC algorithm than for the OPPC algorithm. For $n = 26$ the PC algorithm needs approximately double the CI tests, while for $n = 150$ it is already more than fifteen times the CI tests in the OPPC algorithm. Moreover, we see that there is a direct connection between the needed CI tests in the OPPC algorithm and the number of deleted edges between incompatible nodes. It is shown in Table 1 that for $d = 5$ about one third of the deleted edges has been removed because the nodes were incompatible. With the number of deleted edges we mean, more precisely, the number of pairs of nodes a and b which became nonadjacent during the execution of the OPPC algorithm. We count a deletion as incompatible if it happens (during different iterations) in line 13 of the OPPC algorithm. These additional removals lead to a sparser graph compared to the PC algorithm and thus, to a reduced search space. Of course, some of these edges might have been removed by finding an independence statement with the same order in the PC algorithm, but in total there is a significant influence. In contrast, for $d = 3$ a smaller percentage of deleted edges were due to incompatible nodes. This might

Table 2: The number of CI tests carried out by the PC algorithm and the OPPC algorithm for random DAGs ($p = k/(n-1)$ is the probability that a certain edge is present in the skeleton). The shown values are the median of 50 random trials. OPPC (uq.) counts the number of unique tests a run of the OPPC algorithm executes. On the right we present the number of deleted edges due to incompatible nodes during a run of the OPPC algorithm contrasted with the total number of deleted edges. More precisely, if both $a \to b$ and $a \leftarrow b$ have been removed we count this as one deleted edge.

DAG		Number of CI tests			Deleted edges (OPPC)	
n	k	PC	OPPC	OPPC (uq.)	Edges betw. IN	All edges
25	3	6,191	11,890	3,967	30	263
50	3	25,380	79,464	13,989	149	1,151
75	3	59,856	254,851	27,008	314	2,663
100	3	97,672	600,161	37,315	503	4,800
125	3	107,840	1,158,631	42,384	764	7,565
150	3	259,849	2,043,558	77,208	1,036	10,952
25	4	59,603	40,347	-	39	251
50	4	413,074	251,365	-	227	1,127
75	4	2,103,822	898,320	-	574	2,628
100	4	7,828,627	2,614,204	-	1,013	4,753
25	5	314,133	245,452	-	38	238
50	5	10,532,940	4,806,021	-	269	1,102
75	5	100,443,318	44,502,264	-	718	2,592
25	6	2,160,401	1,761,610	-	31	226
50	6	199,871,850	83,824,159	-	260	1,077

be explained by the fact that a lot of edges are removed immediately through marginal independences.

8.3 Results in the Erdős–Rényi model

In this subsection we test the performance of the OPPC algorithm on DAGs generated with the Erdős–Rényi model. In this graph generation model every (undirected) edge is present independently with probability p. Afterwards, the edges are directed according to a random topological sorting — meaning a random permutation — of the nodes. We will choose p as $k/(n-1)$ with k being the expected node degree. This enables us to compare the results with the previous subsection where we had a fixed node degree d. The main difference compared to the random regular graph model is that the maximal node degree $d_{\max}$ might be significantly larger than the expected node-degree k. This has a direct influence on the number of needed CI tests because as we have analyzed previously (see

Section 3.3) the number of CI tests grows polynomially in $d_{\max}$.

The results of the tests are shown in Table 2. As we have seen in the regular graph model, the PC algorithm performs better when considering very sparse graphs. This is the case for $k = 3$. Again the issue is that the number of CI tests used by the PC algorithm grows slower than $\mathcal{O}(n^3)$ and the additional tests used in the OPPC algorithm outweigh the improvement of earlier edge removals. However, if we only count unique CI tests, the OPPC algorithm needs fewer CI tests than the PC algorithm. Beginning with $k = 4$ the OPPC algorithm outperforms the PC algorithm. As in the regular graph model, the gap between the two algorithms grows with the number of nodes n. This is exemplified for $k = 4$, where there is a factor ≈ 1.5 between the two algorithms for $n = 25$ and a factor ≈ 3 for $n = 100$. This result shows that in particular for computationally very expensive settings with a large number of nodes, the OPPC algorithm is a better choice than the PC algorithm. A growing k on the other hand has a smaller influence on the relative performance of the algorithms provided k is large enough (here larger than three). For example for $n = 50$ and $k = 5$ there is a factor ≈ 2.2 between the algorithms. For $k = 6$ the factor is ≈ 2.4 and thus very similar. However, it would be necessary to compare these values for larger k. But these tests would be computationally very expensive and from a certain point onward infeasible.

When comparing the results to the previous subsection the first thing to note is that the number of CI tests is larger in the Erdős–Rényi model. The reason for this, as mentioned above, is that the the maximal node degree $d_{\max}$ is usually higher than k and that the running time of the algorithms is polynomial in $d_{\max}$. Moreover, the OPPC algorithm outperforms the PC algorithm by a larger margin when considering regular graphs. For the case of $n = 50$ and $d = 6$ there is a factor of ≈ 3.5 between the algorithms while in the Erdős–Rényi model for $n = 50$ and $k = 6$ it is below 2.5. One possible reason might be that as for regular graphs the node degree is fixed and therefore every node has the maximal node degree, deleting an edge due to incompatible nodes anywhere in the graph will have a direct impact on the number of CI tests. In the Erdős–Rényi model on the other hand we might delete an edge between two incompatible nodes with a relatively low degree. The impact of this deletion will probably be significantly lower.

8.4 Analysis of the underlying distributions

The previous subsections gave a good overview of the performance of the OPPC algorithm in comparison with the PC algorithm. In this subsection, we consider one choice of parameters for both DAG generation models in more detail.

For the presentation of the results in the previous subsections, we chose one representative value for each choice of n and d (or p respectively). This value was the median of 50 random trials. We investigate why the use of the median instead of the mean is preferable and in general shed some light on the underlying distribution of the number of needed CI tests in the PC and OPPC algorithm for the used DAG generation models. In order to do

Table 3: 250 executions of the PC and OPPC algorithm were conducted on random DAGs with parameters $n = 76$ and $d = 5$ in the random regular graph model on the left and with parameters $n = 75$ and $p = 5/(n-1)$ in the Erdős–Rényi model on the right. Displayed is the number of trials for which the number of CI tests is in a given range.

No. CI	Occ. PC	Occ. OPPC
$[10^5, 5 \cdot 10^5)$	0	11
$[5 \cdot 10^5, 10^6)$	0	25
$[10^6, 5 \cdot 10^6)$	9	130
$[5 \cdot 10^6, 10^7)$	21	38
$[10^7, 5 \cdot 10^7)$	183	46
$[5 \cdot 10^7, 10^8)$	33	0
$[10^8, 5 \cdot 10^8)$	4	0

No. CI	Occ. PC	Occ. OPPC
$[10^5, 10^6)$	0	2
$[10^6, 10^7)$	12	46
$[10^7, 10^8)$	92	98
$[10^8, 10^9)$	104	71
$[10^9, 10^{10})$	35	29
$[10^{10}, 10^{11})$	7	4

this we will look at the exemplary case of $n = 75$ and $p = 5/(n-1)$ in the Erdős–Rényi model and will compare this to the parameters $n = 76$ and $d = 5$ in the random regular graph model. Note that we are not able to choose $n = 75$ in the latter model as no graph with these properties exists.

In Table 3 the distribution of the number of CI tests in the PC and OPPC algorithm is displayed for 250 random trials. For various ranges we display the number of executions which fall into this range. For example, in the random regular model (on the left) there are 183 executions of the PC algorithm for which the number of CI tests lies between 10^7 and $5 \cdot 10^7$ while there are 46 such executions of the OPPC algorithm. In the Erdős–Rényi model (shown on the right) the number of needed CI tests varies between 10^5 and 10^{11}. For random regular graphs the distribution is denser and for the given experiments it lies completely in the range between 10^5 and $5 \cdot 10^8$. A reason for this is that the maximum degree is fixed in the case of regular graphs while $d_{\max}$ might vary in the Erdős–Rényi model. Recall that the number of CI tests grows with the maximal degree in the PC and OPPC algorithm.

We take a look at the mean and the median for the random regular graph distribution. The means are $29,620,683$ and $5,803,033$ for PC and OPPC algorithm respectively, and the medians are $23,102,446$ and $3,040,522$. In this model the mean and the median lie quite close to each other. In the Erdős–Rényi model, on the other hand, the means are $1,169,373,713$ and $902,187,603$ for PC and OPPC algorithm respectively, and the medians are $148,319,989$ and $60,574,544$. We can see that there is a large difference between mean and median. This applies to the absolute values as well as the relative performance of the two algorithms. We will investigate why this is the case and argue that the median is more suitable than the mean. We do this by comparing single runs of the PC and OPPC algorithm.

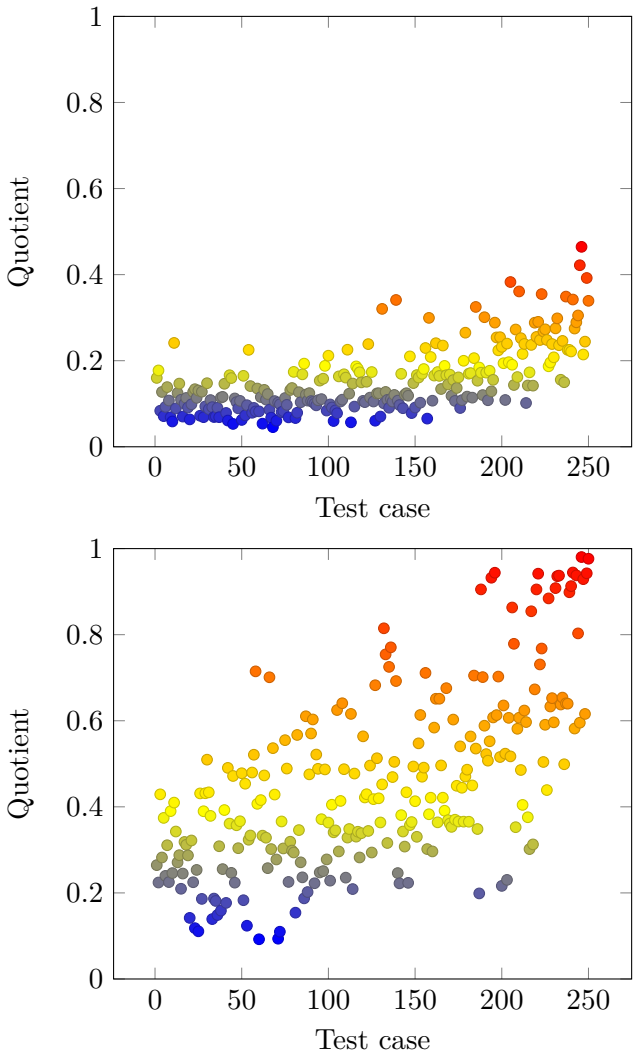

Figure 13: 250 executions of the PC and OPPC algorithm were conducted on random DAGs with parameters $n = 76$ and $d = 5$ in the random regular graph model on the top and with parameters $n = 75$ and $p = 5/(n-1)$ in the Erdős–Rényi model on the bottom. Displayed is the quotient of the needed number of CI tests in the OPPC algorithm and the PC algorithm ($\#\mathrm{CI}_{\mathrm{OPPC}} / \#\mathrm{CI}_{\mathrm{PC}}$). The data are sorted by increasing number of CI tests in the PC algorithm such that the data point with x-coordinate k stands for the input graph for which the PC algorithm uses the k lowest amount of CI tests.

In Figure 13 we display *for each random trial* the quotient of the needed number of CI tests in the OPPC and in the PC algorithm ($\#\mathrm{CI}_{\mathrm{OPPC}} \,/\, \#\mathrm{CI}_{\mathrm{PC}}$). These quotients are presented in an order such that the number of CI tests needed in the execution of the PC algorithm increases. If a quotient is below one the OPPC algorithm needs fewer CI tests, if it is exactly one both algorithm need the same number of tests and if it is above one the PC algorithm needs fewer tests. At first, we take a look at the lower figure which shows the results in the Erdős–Rényi model. The first thing to notice is that the quotient is always below one. This means that the OPPC algorithm is better than the PC algorithm for every test case. However, exactly the cases with the most number of CI tests are the ones where the quotient is closest to one. This is exactly the reason why the mean captures the performance of the OPPC algorithm poorly. It skews the result towards the test cases with a large number of CI tests. But these do not represent the results very well. We see this when comparing the quotient of the means with the quotient of the medians. The former value is ≈ 0.8, while the latter is ≈ 0.4 and, thus, lies much more in the centre of the quotients shown in Figure 13. For the random regular graphs which are presented in the upper figure the same observation holds. The quotient is the largest for the costliest test cases. However, the phenomenon is less extreme. It follows that mean and median differ less extremely as well, but overall the mean is still better suited to capture the relative performances of the two algorithms. Moreover, we see that the quotient is always below 0.5. This confirms our observation that the improvement through the use of the OPPC algorithm is larger for regular graphs.

From the investigations above we can conclude that the performance gains of the OPPC algorithm are larger for the *easier* test cases when considering fixed parameters n and d (or p respectively). In other words, precisely for the hardest test cases it is difficult to significantly improve on the PC algorithm. A possible reason for this observation is the following. If the test case is harder, this could mean that there are certain substructures for which most independences are only found for higher orders and learning this substructure constitutes a large portion of the needed number of CI tests. In this case the earlier deletion of edges between incompatible nodes in the rest of the graph and the unnecessary CI tests which are thereby omitted will be less significant compared to the total number of CI tests. On the other hand, there is not much room for improvement when dealing with these hard substructures, because independence statements are found relatively late. In the extreme case this becomes very clear. Consider the skeleton of the underlying DAG to be a complete graph. In this case no CI will be found and there seems to be no possible shortcut to the exhaustive brute-force search through all possible CIs. This means, in turn, that ideas like the notion of incompatible nodes cannot improve the performance at all. However, it is reasonable to assume that especially such extreme test cases will not be the ones encountered in practice.

We will end the section by investigating how early edges are removed from the skeleton in the PC and OPPC algorithm. It was our goal in the OPPC algorithm to mainly use low-order CIs. We will check if this is indeed the case. Both algorithms work in an iterative

Table 4: Number of edges deleted for a given order l. Precisely, we mean the number of newly nonadjacent nodes. In the random regular graph model we consider the parameters $n = 76$ and $d = 5$, in the Erdős–Rényi model $n = 75$ and $p = 5/(n-1)$. The values are the mean of 250 random trials.

	Regular graph model		Erdős–Rényi model	
Order l	Del. edges PC	Del. edges OPPC	Del. edges PC	Del. edges OPPC
0	992	1,672	964	1,436
1	641	527	597	540
2	434	199	416	240
3	299	117	271	144
4	192	80	168	97
5	102	65	85	57
6	0	0	50	39
7	0	0	21	19
8	0	0	10	10
9	0	0	3	3
10	0	0	1	1
11	0	0	1	1

fashion, first considering marginal independences ($l = 0$), then CIs with order $l = 1$ and so on. In Table 4 we show how many pairs of nodes a and b become nonadjacent during the search for CIs of order l. This number is interesting because it directly influences the search space for possible separating sets Z. For the PC algorithm it is clear that this is equivalent to the number of newly found conditional independences ($a \perp\!\!\!\perp b \mid Z$) for some Z with $|Z| = l$. In the OPPC algorithm there is the additional case of a and b being incompatible nodes. The shown values are the mean of 250 random trials for the parameters $n = 76$ and $d = 5$ in the regular graph model and $n = 75$ and $p = 5/(n-1)$ in the Erdős–Rényi model.

We see that the OPPC algorithm is able to remove much more edges just by considering marginal independences. For example in the regular graph model 992 pairs of nodes are nonadjacent in the PC algorithm after searching through the marginal independences while there are 1,672 pairs of such nodes in the OPPC algorithm. Similarly, in the Erdős–Rényi model there are 964 pairs of nodes which become nonadjacent when considering the marginal independences in the PC algorithm compared to 1,436 pairs in the OPPC algorithm. Thus, the search space for higher orders is significantly reduced which, on the one hand, speeds up the algorithms and, on the other hand, means that we have to test fewer high order CIs. This was exactly the motivation for the OPPC algorithm because especially high-order CIs are difficult to test. On the other hand, we also see that even in

the OPPC algorithm there are certain independences which we only find when testing for high orders. In the Erdős–Rényi model it is necessary for both algorithms to check for CIs with order eleven on certain instances. This confirms our conjecture above that the OPPC algorithm is not able to significantly improve the learning of certain hard substructures. It needs to be further investigated if there is a different approach which is able to speed up the learning of these substructures as well.

9 Discussion

The main goal of this thesis was to exploit low-order CIs in order to improve constraint-based causal structure learning. We started with the problem of recovering causal structures from sets of order-bounded CIs and solved this problem for all sets of conditional independences up to a fixed order k. Moreover, we were able to show that this approach can be immediately applied to causal structure learning in general by proposing the OPPC algorithm which outperforms the popular PC algorithm for various graph classes. However, the research done in this thesis also gives rise to new questions.

While we proposed an algorithm to obtain the graph G which represents the set $\mathcal{F}(\mathcal{I}_V^k)$ of faithful DAGs and which enables us to find all faithful DAGs, we lack an algorithm which enumerates all faithful DAGs. The trivial algorithm we proposed at the end of Section 6 outputs some DAGs multiple times. In order to tackle this problem the study of edge minimal faithful DAGs would be interesting. Textor et al. [18] used the edge minimal faithful DAGs to solve the enumeration problem for the special case of marginal independences. The question arises if their approach can be generalized. In this thesis we mainly focused on edge maximal faithful DAGs and their connection to the representation G. The edge minimal faithful DAGs could also give us a deeper understanding of the structure of $\mathcal{F}(\mathcal{I}_V^k)$.

We have argued that the notion of a representation of a set $\mathcal{I}_V^k$ of CIs up to order k can be viewed as a generalization of the concept of a CPDAG. It would be interesting to apply this connection, for example, to the important open problem of counting Markov equivalence classes [15]. The basis of the research could be the problem of counting the number of representations for sets of marginal independences $\mathcal{I}_V^0$ which was solved by Textor et al. [18].

The OPPC algorithm learns causal structures more efficiently than the PC algorithm. But for very sparse graphs the standard implementation needs more CI tests compared to the PC algorithm due to the relatively large number of redundant tests (at most $\mathcal{O}(n^3)$). We showed that when only considering unique CI tests the algorithm is better than the PC algorithm. However, an implementation which, for example, stores the already tested CIs in order to avoid redundant tests is clearly not in the spirit of the algorithm and would lead to a much increased memory demand. Thus, the question remains how to implement this algorithm efficiently in a way which avoids these redundant tests.

Apart from this an analysis of the statistical properties of the OPPC algorithm would be of interest. This lies outside the scope of this thesis as we solely focus on learning from sets of CIs and not directly from data. In combination with this, experimental tests on generated or even real world data are needed to analyze how well the OPPC algorithm works in practice.

References

[1] Joaquín Abellán, Manuel Gómez-Olmedo, and Serafín Moral. Some variations on the PC Algorithm. In *Probabilistic Graphical Models*, 2006.

[2] 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, 04 1997.

[3] Wicher P. Bergsma. Testing conditional independence for continuous random variables. *Report Eurandom*, 2004.

[4] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5(Oct):1287–1330, 2004.

[5] Diego Colombo and Marloes H. Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15(1):3741–3782, 2014.

[6] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[7] Luis M. De Campos and Juan F. Huete. A new approach for learning belief networks using independence criteria. *International Journal of Approximate Reasoning*, 24(1):11–37, 2000.

[8] Markus Kalisch and Peter Bühlmann. Estimating high-dimensional directed acyclic graphs with the PC-Algorithm. *Journal of Machine Learning Research*, 8:613–636, May 2007.

[9] Markus Kalisch and Peter Bühlmann. Robustification of the PC-algorithm for directed acyclic graphs. *Journal of Computational and Graphical Statistics*, 17(4):773–789, 2008.

[10] 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.

[11] Junning Li and Z. Jane Wang. Controlling the false discovery rate of the association/causality structure learned with the PC algorithm. *Journal of Machine Learning Research*, 10(Feb):475–514, 2009.

[12] Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 403–410. Morgan Kaufmann Publishers Inc., 1995.

[13] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2nd edition, 2009.

[14] Judea Pearl and Nanny Wermuth. When can association graphs admit a causal interpretation? In *Selecting Models from Data*, pages 205–214. Springer, 1994.

[15] Adityanarayanan Radhakrishnan, Liam Solus, and Caroline Uhler. Counting markov equivalence classes by number of immoralities. In *Proceedings of the Thirtythird Conference on Uncertainty in Artificial Intelligence*, UAI '17, 2017.

[16] Ross D. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 480–487. Morgan Kaufmann, 1998.

[17] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT press, 2nd edition, 2000.

[18] Johannes Textor, Alexander Idelberger, and Maciej Liśkiewicz. Learning from pairwise marginal independencies. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 882–891. AUAI Press, 2015.

[19] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 255–270. Elsevier Science Inc., 1991.

[20] Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proceedings of the Eighth International Conference on Uncertainty in Artificial Intelligence*, UAI'92, pages 323–330. Morgan Kaufmann Publishers Inc., 1992.

[21] Anja Wille and Peter Bühlmann. Low-order conditional independence graphs for inferring genetic networks. *Statistical applications in genetics and molecular biology*, 5(1), 2006.