Technical University of Munich

Department of Mathematics

Cluster DAGs as Background Knowledge for Causal Discovery

Master's Thesis

of

Jan Marco Ruiz de Vargas Staudacher

Supervisor:	Prof. Dr. Niki Kilbertus
Advisor:	M.Sc. Kirtan Padh
Submission Date:	30th November 2023

I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Jan Marco Ruiz de Vargas Staudacher

Aschheim, 30.11.2023

Acknowledgements

I would like to thank Professor Niki Kilbertus for the opportunity to write my thesis under his supervision. I am especially grateful that he allowed me to choose a topic I suggested and let me explore the field very freely. The discussions we had deepened my understanding of the material and motivated me greatly to dive deeper into the research literature. His insightful advice was a great guidance for me.
Furthermore I would also like to thank Kirtan Padh for his support, feedback and time. Our conversations helped me gain a new perspective and expanded my knowledge on the subject.
I am also deeply thankful to my parents for their ongoing support in my years of study and the opportunities they provided me. My father first introduced me to causality, which motivated me to pursue it in my studies. Last but not least, I would like to thank my sister, my friends and my wonderful girlfriend Muchun for their companionship, their encouragement and the fun we had together during leisure activities.

German Abstract

Causal discovery (kausale Entdeckung) ist eine wichtige Aufgabe in der wissenschaftlichen Forschung und im maschinellen Lernen. Sie zielt darauf ab, einen Graphen aus Daten schätzen, der Ursache-Wirkungs-Beziehungen visualisieren kann. State of the Art Methoden haben Schwierigkeiten mit hochdimensionalen Daten und nicht spärlichen Abhängigkeiten. Oft steht Vorwissen über das zu schätzende System zur Verfügung. Dies führt zur Idee des Hintergrundwissens (background knowledge), das vorhergehendes Wissen über den Graphen beschreibt.
Viele Varianten des Hintergrundwissens wurden vorgeschlagen, in dieser Arbeit konzentriere ich mich auf Cluster gerichtete azyklische Graphen (C-DAGs), die eine intuitive und leicht lesbare Möglichkeit bieten, Variablen aufgrund von angenommenen Beziehungen zwischen Gruppen von Variablen zusammenzufassen. Ich zeige, dass C-DAGs als eine boolesche Kombination von paarweisem Hintergrundwissen ausdrückbar sind, was sie auf eine solide theoretische Grundlage stellt. Darüber hinaus stelle ich verschiedene Hintergrundwissens-Frameworks vor und vergleiche sie mit C-DAGs. C-DAGs zeigen vorteilhafte Eigenschaften gegenüber anderen Hintergrundwissens-Frameworks, die nützliche Einschränkungen aber auch Flexibilität bei der Modellierung von Gruppenbeziehungen ermöglichen.
Weiterhin zeige ich, dass C-DAGs als Werkzeug für kausale Entdeckung genutzt werden können. Nach meinem besten Wissen ist dies die erste Arbeit, die C-DAGs in diesem Kontext verwendet. Ich habe hierfür ein Python-Paket namens clustercausal entwickelt. Im Gegensatz zur Mehrheit der verfügbaren Literatur, die Vorwissen als ein Nachverarbeitungswerkzeug verwendet, werden C-DAGs verwendet um den kausalen Entdeckungsalgorithmus selbst zu verbessern. Dies führt zum Cluster-PC Algorithmus, der die C-DAG Struktur verwendet um den populären PC Algorithmus zu verbessern. Simulationen zeigen, dass dies die Anzahl der notwendigen Bedingte-Unabhängigkeitstests signifikant verringert und den Algorithmus schneller macht. Darüber hinaus übertrifft Cluster-PC in den Simulationen in der Regel die Baseline-Version des PC-Algorithmus über eine große Variation von Hyperparametern hinweg, was darauf hindeutet, dass Cluster-PC eine robuste Verbesserung ist. Zusätzlich führe ich den Cluster-FCI Algorithmus ein, der in der Lage ist, latente Variablen zu modellieren. Dieser baut auf dem FCI Algorithmus auf und nutzt das C-DAG ähnlich wie Cluster-PC. Auch dieser zeigt in ersten Experimenten vielversprechende Ergebnisse. Sowohl Cluster-PC als auch Cluster-FCI sind in clustercausal implementiert und können dort mit Simulationsstudien untersucht werden.

English Abstract

Causal discovery is an important task in scientific research and machine learning. It aims to recover a graph from data that is able to visualize cause-effect relationships. State of the art methods struggle with high-dimensional data and non-sparse dependencies. Often some prior knowledge about the system to be estimated is available. This leads to the notion of background knowledge, which encodes previous beliefs about the graph. Numerous variants of background knowledge have been suggested, in this thesis I focus on cluster directed acyclic graphs (C-DAGs), which allow an intuitive and easily readable way of grouping variables together based on believed relationships between groups of variables. I show that C-DAGs are expressable as a boolean combination of pairwise background knowledge, which puts them on sound theoretical footing. Furthermore I present different background knowledge frameworks and compare them with C-DAGs. C-DAGs are shown to have advantageous properties over the other background knowledge frameworks, allowing for useful restrictions but also flexibility when modeling group relationships.

Additionally, I show that C-DAGs can be used as a tool for causal discovery. To the best of my knowledge, this is the first work that uses C-DAGs in this context. For this I developed a python package called clustercausal. In contrast to the majority of available literature that uses prior knowledge as a post-processing tool, C-DAGs are used to enhance the causal discovery algorithm itself. This leads to the Cluster-PC algorithm, which uses C-DAG structure to improve the popular PC algorithm. Simulation studies show that this decreases the number of necessary conditional independence tests significantly, making the algorithm faster. In addition, in the simulation studies Cluster-PC generally outperforms the baseline PC version across a big variation of hyperparameters, suggesting that Cluster-PC is a robust improvement. I also introduce the Cluster-FCI algorithm, which is capable of modeling latent variables. It builds on the FCI algorithm and uses the C-DAG similarly to Cluster-PC. This too shows promising results in initial experiments. Both Cluster-PC and Cluster-FCI are implemented in clustercausal and can be examined there with simulation studies.

Contents

1. Introduction

Causal discovery allows one to find cause-effect relationships from observational data and is becoming increasingly important for scientific research. Hence more and more effort has been put into the development of algorithms that can use data to estimate graphs, which are used to represent the dependency structure of a physical system or statistical model. Different types of algorithms have been proposed, popular ones include the constraint based PC algorithm [Spi+00], score based methods like greedy equivalence search (GES, [Chi02]), as well as hybrid methods [MKB09], discovery via structural assumptions [Shi14] and more recently methods using continuous optimization like DAGs with NO TEARS [Zhe+18].

The data for these algorithms is often observational, i.e. data that has been gathered passively without influencing the observed system. This is in contrast to interventional (also called experimental) data, which is usually very costly to obtain or the experiments are unethical. Unfortunately, observational data only allows limited estimation of cause-effect relationships, as statistical tests estimate correlation, not causation. For example, consider a two variable case with sunshine S and ice cream consumption I. An observational study can only conclude that S and I are correlated, but not the direction in which causation flows. To determine that direction, one could perform an experiment. Get everyone to eat ice cream on a rainy day and observe whether the sun comes out or not. It does not, so the graph encoding the causal structure, also called a directed acyclic graph (DAG), must be $S \to I$.

The effort and expenses required to conduct such an experiment would be substantial, and this challenge extends beyond simple example problems. And of course this experiment is nonsensical, as anyone with their previous knowledge about the world can easily argue that it should be $S \to I$, because the warm weather makes people crave for ice cream, while people eating ice cream cannot affect a stellar object 150 million kilometres away.

This previous knowledge is also called background knowledge and has been a key topic of interest for improving causal discovery in recent years. In the aforementioned example, the background knowledge alleviates the need for an experiment to determine causal direction. Causal discovery also suffers from other problems that background knowledge can ameliorate. First, when faced with high-dimensional and non-sparse graphs, the runtime of e.g. the PC algorithm is exponential in the number of vertices [Le+16] because the space of possible DAGs grows super exponential in the number of vertices and therefore search becomes slow for large graphs [KLC22]. Second, the data itself may be misleading or assumptions of the model might not be met. A conditional independence test (CI test), which can be used to determine presence or absence of edges in a graph, may return a false result, which leads to a wrong graph. Lastly, those methods cannot always return a DAG. Usually they return an equivalence class of DAGs, as only that is determinable from conditional independence structure alone. This Markov equivalence class (cf. Definition 2.13) might be too large to derive useful causal information from.

Therefore enhancing these algorithms represents a compelling area of interest within the field of causal discovery. As alluded to before, in practice researchers are often faced with the situation of partial information: they possess some domain knowledge and can rule out certain DAG structures, but they aren't certain enough to construct a DAG with 100% confidence. Numerous types of background knowledge have been introduced ([CGK23] for an overview and [ASC20], [Fan+22], [Bro+22], [HG23], [Ana+23] for particular methods). The idea is to represent the prior knowledge as a restriction on the graph or the search space and use that to either guide the algorithm ([KLC22], [ASC20]) or prune its result ([Bro+22], [WQZ22]). This benefits causal discovery in all three aforementioned problems:

- Restricting the graph via prior knowledge reduces the search space and hence the computational load for the discovery algorithm.
- Structure defined by prior knowledge does not have to be statistically tested again, making the algorithm less prone for errors.
- Additional arrow directions from the background knowledge reduce the size of the Markov equivalence class ([Bro+22], [BD23]).

Of particular interest are Cluster-DAGs (C-DAGs, [Ana+23]). This background knowledge framework was introduced for use in causal inference. C-DAGs group variables into clusters and specify relationships only between those clusters. C-DAGs then allow do-calculus and adjustment on the cluster macro variables [Ana+23]. C-DAGs also provide a very intuitive and easy to read way of representing background knowledge, which makes them interesting for applications in causal discovery.

Consider the hypothetical example of a healthcare professional examining causes of burnout. The hypothetical DAG for the variables that the researcher measures is given in Figure 1.1. The researcher only has access to data collected on these variables and does not know about the ground truth DAG of Figure 1.1. A causal discovery algorithm like PC would return an undirected graph in this case (due to the absence of v-structures, cf. Definition 2.2 (v)), shown in Figure 1.2.

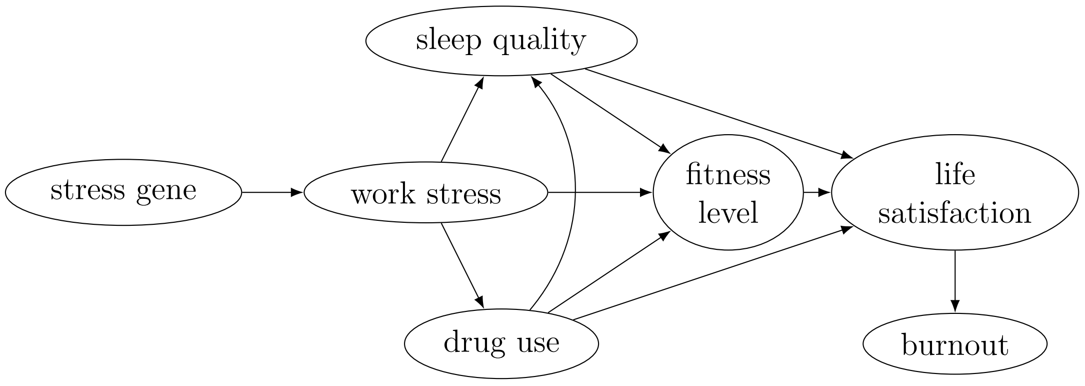

Figure 1.1.: Hypothetical burnout example.

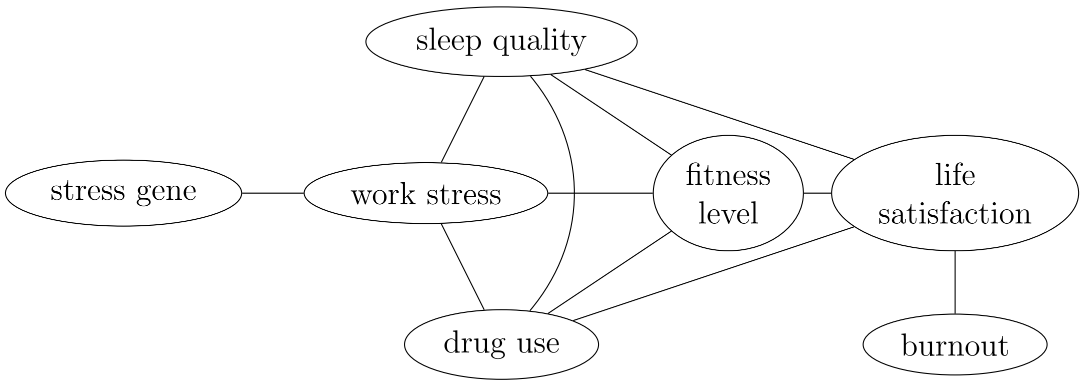

Figure 1.2.: Output of PC algorithm for example in Figure 1.1.

Unfortunately the graph in Figure 1.2 does not answer the question whether life satisfaction influences burnout or the other way around, as the undirected edge between them represents ambiguity about the causal direction.

Here a C-DAG can help: the researcher knows that she can group variables into two clusters, $C_1 = \{\textit{stress gene}\}$ and $C_2 = \{\textit{work stress}, \textit{sleep quality}, \textit{fitness level}, \textit{drug use}, \textit{life satisfaction}, \textit{burnout}\}$ with C-DAG $C_1 \to C_2$ because the genes of a person are determined before all other variables exist. With this added knowledge, the edge *stress gene* $\to$ *work stress* can be oriented. Furthermore, due to edge orientation rules about v-structures, *stress gene* $\to$ *work stress* $-$ *sleep quality* can be oriented to *stress gene* $\to$ *work stress* $\leftarrow$ *sleep quality*. In the same way, many more edges can be oriented, leading to the graph in Figure 1.3, which is then also able to answer that *life satisfaction* causes *burnout*, show the causes of *life satisfaction* and more.

This example shows the usefulness of C-DAGs for causal discovery. To the best of my knowledge, C-DAGs have not been used for causal discovery before, which is the core topic of this thesis. In Chapter 2 I recap the preliminaries on graphical statistical models and causal discovery which will be needed later on. Chapter 3 gives a review

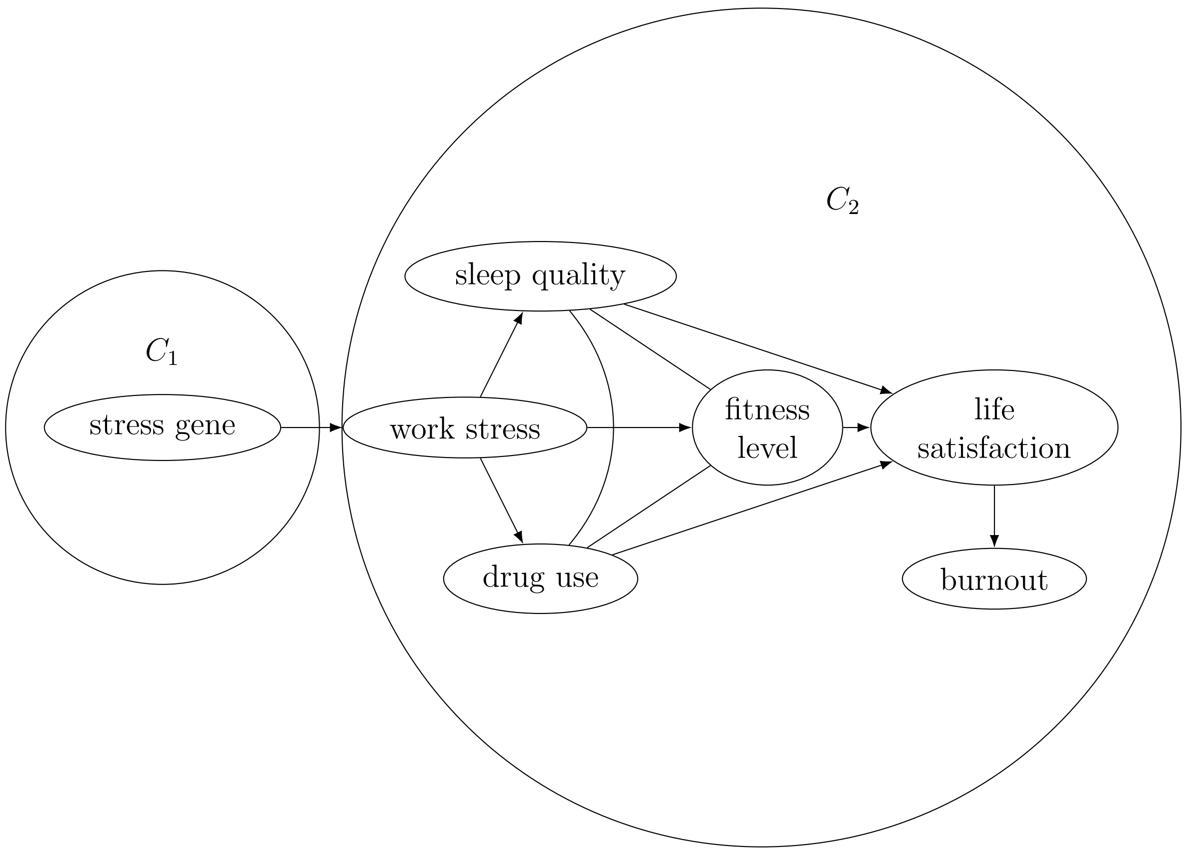

Figure 1.3.: C-DAG allows for additional edge orientations

of background knowledge frameworks and compares C-DAGs with related frameworks, highlighting and discussing their differences and advantages. In particular, I show how C-DAGs are expressable as a boolean form of pairwise background knowledge, leading to better comparisons across background knowledge frameworks and faster transfer of theoretical results. Chapter 4 presents the Cluster-PC algorithm, my adapted version of the PC algorithm. By adapting the C-DAG structure to warm-start and guide the algorithm, in addition to being able to orient more edges, Cluster-PC is more efficient in the number of conditional independence tests and is potentially able to avoid erroneous conditional independence tests. The results of the simulation studies strengthen these claims. I also show soundness and completeness of Cluster-PC. Chapter 5 demonstrates how to handle latent variables with C-DAGs and introduces the corresponding Cluster-FCI algorithm. Soundness of Cluster-FCI is shown. In Appendix A, a short guide on the usage of the clustercausal package is presented, which I developed for this thesis. It implements the Cluster-PC, the Cluster-FCI algorithm and allows the execution and evaluation of large simulation studies. Chapter 6 concludes and discusses potential further research questions.

2. Preliminaries on causal models and causal discovery

This chapter is an introduction to causality and the mathematical framework of graphical statistical models that is needed for the rest of this thesis. Section 2.1 presents how graphs connect to distributions so that they can be used as statistical models. Notation for directed acyclic graphs (DAGs) and node relations, d-separation in particular, is introduced. The definition of the global Markov property and faithfulness allow one to relate graphs to distributions precisely. Furthermore I characterize Markov equivalence (cf. Definition 2.13), which shows that different DAGs may entail the same conditional independencies, making them Markov equivalent. Markov equivalence is important for causal discovery, as it implies that only an equivalence class of DAGs can be recovered from data. An important assumption also introduced in Section 2.1 is causal sufficiency (cf. def 2.17). Section 2.2 expands the previous framework to the situation when the assumption of causal sufficiency is not met. This means modeling of latent variables, i.e. variables that are influencing the observerd variables, but are not observed themselves. A new type of graph, the acyclic directed mixed graph (ADMG), and a new separation criterion, m-separation, generalize DAGs and d-separation for this task. ADMGs are relevant later on for the discussion of C-DAGs in Section 3.1.

Section 2.3 gives a primer in causal discovery and introduces the main types of algorithms used, namely constraint based algorithms, score based algorithms, continuous optimization methods and discovery via structural assumptions. I present the constraint based PC algorithm [Spi+00] and FCI algorithm ([Spi+00], [Zha08b]) in detail in Sections 2.3.1 and 2.3.3 respectively. For the PC algorithm I introduce the target output graph which represents the Markov equivalence class, the completed partially directed graph (CPDAG). I state the PC algorithm and discuss ways it has been optimized in recent literature. For the FCI algorithm, which is able to model latent variables, I introduce a new class of graphs in Section 2.3.2. I show that ADMGs are not the correct type of graph to use for causal discovery, therefore ancestral graphs are defined. Ancestral graphs may contain nodes that are not adjacent, but also not m-separable. To work around this issue, inducing paths and the related maximally ancestral graphs (MAGs) are defined. I also show how a DAG can be transformed into a MAG. Lastly I present what Markov equivalence means for MAGs, which leads to the target output of the FCI algorithm, the partial ancestral graph (PAG). Section 2.3.3 shows the FCI algorithm in detail and discusses some similarities and differences with respect to the PC algorithm. The PC and FCI algorithms will be expanded upon later with the inclusion of

C-DAG background knowledge in Chapters 4 and 5. Section 2.4 concludes with the preliminaries.

2.1. Graphs as statistical models

Graphs are a powerful tool for statistical modeling. They go back to Sewall Wright in the early 20th century ([Wri18], [Wri34]), who introduced path coefficients to model correlation coefficients flexibly. Their core idea is to relate a graph to a probability distribution, with nodes corresponding to random variables and edges corresponding to correlations. In this context, there are two types of graphs commonly used: undirected graphs and directed graphs. For undirected graphs, a missing edge between nodes means there exists some (conditional) independence between those nodes, while an edge indicates that there exists no (conditional) independence between them. [HTW15] Chapter 9 goes into more details on undirected graphical models.

Directed graphs, on whom I focus in this thesis, build on undirected ones. The arrowheads are interpreted to show the direction of an effect. If A, B are random variables with joint distribution $P(A, B)$, the arrow $A \rightarrow B$ indicates that there is some physical mechanism such that a change in A will influence B, but not the other way around. One sees how this leads to a causal framework, using directed graphs to reason causally about the probability distribution. Note that there is another school of thought on modeling causality, the framework of potential outcomes [Rub05]. [RR13] presents a way to unify these two frameworks using single world intervention graphs (SWIGs). This thesis however follows the Pearlian [Pea09] model of causality, with structural causal models (SCMs) and directed acyclic graphs (DAGs), as this framework is very popular and most work on causal discovery follows its approach.

The introduction to the definitions, theorems and notations follows [Kil22], an unpublished causality lecture script at TUM from the year 2022. Proofs of well-known statements that do not concern this thesis directly may be omitted and can be found in the cited papers or books on causality such as [PJS17], [Pea09] or [Lau96].

Directed graphs are a tupel of nodes and edges, where edges can also have a direction, either 'left' $\leftarrow$ or 'right' $\rightarrow$. If direction of an edge is unclear, it is denoted as the undirected edge $-$.

Definition 2.1 (Directed graph). *Let V be a set. A tuple $G := (V, E)$ is called a directed graph (or just graph for short) with V the set of nodes and edges $E \subset V \times V$, $(v, v) \notin E \; \forall v \in V$.*

Often I will shorten the notation $G = (V, E)$ to G when talking about a graph when the nodes and edges are clear from context. For examples the 'generic' node set $V = \{X_1, ..., X_d\}$ or nodes X, Y, Z will be used. Nodes in a graph can relate to each other in a variety of ways:

Definition 2.2 (**Node relations**). *Let $G = (V, E)$ be a graph, $X_i, X_j \in V$, $X_1, ..., X_n \in V$ and $n \geq 2$.*

(i) Nodes X_i, X_j are adjacent if $(X_i, X_j) \in E$ or $(X_j, X_i) \in E$. All nodes adjacent to X_i are called the neighbors $nb^G_{X_i} := \{X_j \mid (X_i, X_j) \in E \text{ or } (X_j, X_i) \in E\}$. The set of neighbors $nb^G_{X_i}$ may also be denoted as the node of adjacent vertices, i.e. $adj^G_{X_i}$.

(ii) A node X_i is called a parent of X_j if $(X_i, X_j) \in E$ and $(X_j, X_i) \notin E$ and a child if $(X_j, X_i) \in E$ and $(X_i, X_j) \notin E$. The set of parents of X_j is denoted as $pa^G_{X_j}$ and the set of children as $ch^G_{X_j}$.

(iii) A node without parents is called a source node, a node without children is called a sink node.

(iv) There is an undirected edge between X_i and X_j if $(X_i, X_j), (X_j, X_i) \in E$, which I denote with $X_i - X_j$. An edge $(X_i, X_j) \in E$ is called directed if it is not undirected. This is then written as $X_i \rightarrow X_j$ for $(X_i, X_j) \in E$.

(v) Three nodes form a v-structure if $X_i \rightarrow X_j \leftarrow X_k$ and X_i, X_k are not adjacent (v-structures are also called unshielded colliders or immoralities).

(vi) Three nodes form a fork if $X_i \leftarrow X_j \rightarrow X_k$ and X_i, X_k are not adjacent.

(vii) A path in G is a sequence of distinct nodes $\{X_1, ..., X_n\}$ ($n \geq 2$) such that X_k, X_{k+1} are adjacent for $k \in \{1, ..., n-1\}$. A path is called directed if $X_k \rightarrow X_{k+1}$ for all k. Paths can be denoted by either writing out their edges like $X_1 - X_2 \rightarrow X_3$ or stating the ordering of the nodes on the path, i.e. (X_1, X_2, X_3).

(viii) A node j is a descendant of X_i if there exists a directed path from X_i to X_j. Then X_i is an ancestor of X_j. The sets of all descendants of X_j are denoted as $de^G_{X_j}$ and the set of all ancestors of X_i is denoted as $an^G_{X_i}$. Node X_j is a non-descendant of node X_i if $X_i \neq X_j$ and X_j is not a descendant of X_i. The set of non-descendants of X_j is denoted as $nd^G_{X_j}$.

(ix) If $X_{k-1} \rightarrow X_k \leftarrow X_{k+1}$ in a path, X_k is called a collider on this path.

Sometimes the suffix G (e.g. in pa^G_X) is dropped out of convenience when the graph is clear from context. By convention, a node X_i is neither a descendant, ancestor or non-descendant of itself. I also introduce the following properties of graphs and special types of graphs that will be useful:

Definition 2.3 (**Graph properties, PDAG, DAG**). *Let $G = (V, E)$ be a graph.*

(i) A graph $G' = (V', E')$ is called a subgraph of G, denoted by $G' \leq G$ if $V' = V$ and $E' \subset E$. If additionally $E' \neq E$, G' is called a proper subgraph of G, denoted by $G' < G$. If G' is a subgraph of G, then G is called a supergraph of G'.

(ii) A graph G is called fully connected if all pairs of nodes are adjacent.

(iii) The skeleton of G is a graph $G' = (V, E')$, where $(X_i, X_j) \in E'$ if $(X_i, X_j) \in E$ or $(X_j, X_i) \in E$. The skeleton is a fully undirected graph.

(iv) A graph G is called a partially directed acyclic graph (PDAG) if there is no directed cycle, i.e., no pair of nodes X_i, X_j with directed paths from X_i to X_j and from X_j to X_i.

(v) A PDAG G is called a directed acyclic graph (DAG) if all edges are directed.

Since directed graphs are meant to represent probability distributions where conditional independence is a notion of separation, one would also like to have such a feature in directed graphs. This is given by d-separation.

Definition 2.4 (**Blocked path**). *In a DAG G, a path between nodes X_1 and X_n is blocked by a set S with $X_1, X_n \notin S$ if there is a node X_k such that either*

(i) $X_k \in S$ and $X_{k-1} \to X_k \to X_{k+1}$ or $X_{k-1} \leftarrow X_k \leftarrow X_{k+1}$ or $X_{k-1} \leftarrow X_k \to X_{k+1}$,

(ii) $X_{k-1} \to X_k \leftarrow X_{k+1}$ and neither X_k nor any of its descendants is in S.

A path that is not blocked by a set S is called active given S. Active paths are also called open and blocked paths are also called closed (with respect to a set S).

Definition 2.5 (**d-separation**). *Let G be a graph. Two subsets A, $B \subset V$ are d-separated by $S \subset V$, if all paths between $X_i \in A$ and $X_j \in B$ are blocked by S. I write $A \perp\!\!\!\perp_G B \mid S$.*

One usually assumes a one-to-one correspondence between nodes $V = \{X_1, \dots, X_d\}$ and random variables $X = (X_1, \dots, X_d)$, hence the same notation. This means X_i can refer to the random variable or the node in the graph, depending on the context.

A distribution can factorize according to the graphical structure of a DAG:

Definition 2.6 (**Factorization According to a DAG**). *A joint distribution P_X factorizes according to a DAG G if*

$$P_X = \prod_{j=1}^{d} P(X_j \mid pa_{X_j}^G) \quad \text{or} \quad p(x) = \prod_{j=1}^{d} p(x_j \mid pa_{X_j}^G). \tag{2.1}$$

Definition 2.7 (**Topological Ordering**). *A permutation $\sigma \in S_d$ is called a topological ordering of the nodes of a DAG G, if*

$$X_j \in de_{X_i}^G \Rightarrow \sigma(i) < \sigma(j). \tag{2.2}$$

When the nodes are ordered according to their topological ordering,
i.e. $V = \{X_{\sigma^{-1}(1)}, ..., X_{\sigma^{-1}(d)}\}$, I also refer to the ordered set V as a topological ordering. For simplicity I often assume that the variables are already topologically ordered, i.e. $\sigma^{-1}(i) = i$ for all i. Due to acyclicity, every DAG has a topological ordering:

Proposition 2.8 (**Existence of Topological Ordering**). *For each DAG, there exists a topological ordering.*

Proof. Every DAG contains at least one source node (who has no ancestors). It can be found by selecting one node and creating a path upward, one parent up each time. As the graph is acyclic and the set of nodes is finite, eventually this path stops at a node without ancestors. This node X_i will be the first in the topological ordering. Afterwards, one repeats the procedure for the graph where the node X_i was deleted. This procedure iteratively adds nodes to the topological ordering, ends after $|V|$ steps and is performable for any DAG. □

In general, the topological ordering is not unique, for example in the DAG $X_1 \to X_3 \leftarrow X_2$, $\{X_1, X_2, X_3\}$ and $\{X_2, X_1, X_3\}$ would both be topological orderings.

Thinking about factorization, the more 'connected' a DAG is, the easier it is for a distribution to factorize on it. A distribution factorizes to all possible fully connected DAGs via the trivial factorization $P_X = P_{X_{\sigma(1)}} P_{X_{\sigma(2)}|X_{\sigma(1)}} P_{X_{\sigma(3)}|X_{\sigma(1)},X_{\sigma(2)}} \cdots P_{X_{\sigma(d)}|X_{\sigma(1)},\dots,X_{\sigma(d-1)}}$.

Next comes the Bayesian network, the object that connects a graph to a probability distribution.

Definition 2.9 (**Bayesian Network**). *A Bayesian network (or DAG model) is a tuple (G, P_X), where G is a directed acyclic graph (DAG) and P_X is a joint distribution that factorizes according to G.*

Conditional independence possesses some properties, whom for completeness are all presented below. (C2) in particular will be needed in Chapter 3.

Theorem 2.10 (**Properties of conditional independence**). *Conditional independence has the following properites for sets of variables X, Y, W, Z:*

(C1) "Symmetry": $X \perp\!\!\!\perp Y \mid Z \Leftrightarrow Y \perp\!\!\!\perp X \mid Z$.

(C2) "Decomposition": $X \perp\!\!\!\perp Y \mid Z \Rightarrow h(X) \perp\!\!\!\perp Y \mid Z$ for any measurable function h. In particular, $(X, W) \perp\!\!\!\perp Y \mid Z \Rightarrow X \perp\!\!\!\perp Y \mid Z$.

(C3) "Weak union": $X \perp\!\!\!\perp Y \mid Z \Rightarrow X \perp\!\!\!\perp Y \mid (Z, h(X))$ for any measurable function h. In particular, $(X, W) \perp\!\!\!\perp Y \mid Z \Rightarrow X \perp\!\!\!\perp Y \mid (W, Z)$.

(C4) "Contraction": $X \perp\!\!\!\perp Y \mid Z$ and $X \perp\!\!\!\perp W \mid (Y, Z) \Leftrightarrow X \perp\!\!\!\perp (W, Y) \mid Z$.

(C5) "Intersection axiom": Suppose there is a joint density $f(x, y, w, z)$ with respect to $\lambda = \lambda_X \otimes \lambda_Y \otimes \lambda_W \otimes \lambda_Z$ such that the marginal $f(y, w, z) > 0$ almost everywhere. Then $X \perp\!\!\!\perp (W, Y) \mid Z \Leftrightarrow X \perp\!\!\!\perp W \mid (Y, Z)$ and $X \perp\!\!\!\perp Y \mid (W, Z)$.

Proof. See [Lau96] Chapter 3. □

For the graph to be a good 'view' of the probability distribution, its edge structure should relate to conditional independence statements. If A and B are d-separated in the graph, then it should also be possible to 'separate' A and B given C in the probability distribution, i.e. them being conditionally independent given C. If that holds true for all sets A, B, C, the global Markov property holds. It is equivalent to the factorization property (and the local Markov property), if a density exists (cf. Theorem 2.12).

Definition 2.11 (**Markov property**). *Let G be a directed acyclic graph (DAG) and P_X a joint distribution. The distribution P_X satisfies*

(i) the local Markov property with respect to G if

$$X_j \perp\!\!\!\perp nd^G_{X_j} \mid pa^G_{X_j} \quad \text{for all} \quad j \in V. \tag{2.3}$$

(ii) the global Markov property with respect to G if

$$A \perp\!\!\!\perp_G B \mid C \Rightarrow A \perp\!\!\!\perp B \mid C \quad \text{for all disjoint sets} \quad A, B, C \subset X. \tag{2.4}$$

(iii) the Markov factorization property with respect to G if

$$p(x) = \prod_{j=1}^{d} p(x_j \mid pa^G_{X_j}). \tag{2.5}$$

Theorem 2.12 (**Equivalence of markov properties**). *All three Markov properties in Definition 2.11 are equivalent and if they hold I say that P_X is Markov w.r.t. G. Existence of a densitiy is required.*

Proof. From [Kil22].
$(i) \Rightarrow (iii)$: Assume without loss of generality that the nodes V are topologically ordered (relative to G). Consider the trivial factorization
$P_X = P_{X_{\sigma(1)}} P_{X_{\sigma(2)}|X_{\sigma(1)}} P_{X_{\sigma(3)}|X_{\sigma(1)},X_{\sigma(2)}} \cdots P_{X_{\sigma(d)}|X_{\sigma(1)},\ldots,X_{\sigma(d-1)}}$ with $\sigma(X_j) = j$ for all $X_j \in V$ and consider one of the factors $p(x_j|x_1, \ldots, x_{j-1})$. According to (i), $X_j \perp\!\!\!\perp nd^G_{X_j} \mid pa^G_{X_j}$ in $p(x)$. Because V is topologically ordered $pa^G_{X_j} \subset \{X_1, \ldots, X_{j-1}\}$ and $de^G_{X_j} \cap \{X_1, \ldots, X_{j-1}\} = \emptyset$. Hence, $\{X_1, \ldots, X_{j-1}\} = pa^G_{X_j} \cup Z$ for some $Z \subset nd^G_{X_j}$. Therefore, the factor in the trivial factorization becomes $p(x_j|x_1, \ldots, x_{j-1}) = p(x_j|pa^G_{X_j})$ by the local independence (and rules of conditional independencies).
$(iii) \Rightarrow (ii)$: This most technical part is omitted for brevity and can be found in [Lau96] Theorem 3.27 or in [KF09] Section 4.5.1.1.
$(ii) \Rightarrow (i)$: Consider $A = \{X_j\}$, $B = nd^G_{X_j}$, and $C = pa^G_{X_j}$. First, note that $A \cap B = A \cap C = \emptyset$ by definition. I need to show that every path from any node $X_i \in B$ to X_j is blocked by C. If $X_i \in pa^G_{X_j}$ there is nothing to show. Otherwise, this is given by the fact that if X_i and X_j are not adjacent and $X_i \in nd^G_{X_j}$, they can be d-separated by $pa^G_{X_j}$ (cf. Proposition 2.27). □

Definition 2.11 is from the perspective of P: P satisfies the lokal Markov property/ global Markov property/ Markov factorization property w.r.t. G. In the following these may be used interchangeably, i.e. P is Markov w.r.t. G or G is Markov w.r.t. P.

Unfortunately, a set of d-separation statements does not in general imply exactly one graph. Different graphs can share the same d-separation structure. With respect to the global Markov property, this means that there may exist several graphs to which a distribution P is Markov. Markov equivalence characterizes this.

Definition 2.13 (**Markov Equivalence**). *Define*

$$M(G) := \{P \mid P \textit{ satisfies the global Markov property with respect to } G\}. \tag{2.6}$$

Two DAGs G_1, G_2 are Markov equivalent if $M(G_1) = M(G_2)$. The set of all DAGs that are Markov equivalent to G is called the Markov equivalence class of G. It can be represented by a completed PDAG $G^* = (V, E')$ (cf. Definition 2.25) , where an edge (i, j) is in E' if and only if it is in any graph of the Markov equivalence class.

Theorem 2.14 (**Characterization of markov equivalence**). *Two DAGs are Markov equivalent if and only if they have the same skeleton and the same v-structures.*

Proof. Can be found in [Pea09]. □

The reverse direction of the global Markov property is called faithfulness. Here, an independence in the distribution implies a separation in the graph.

Definition 2.15 (**Faithfulness**). *The joint distribution P_X is said to be faithful to the DAG G if*

$$A \perp\!\!\!\perp_G B \mid C \Leftarrow A \perp\!\!\!\perp B \mid C$$

holds for all disjoint sets A, B, $C \subset X$.

Definition 2.16 (**Causal minimality**). *Let G be a DAG. A distribution P_X satisfies causal minimality with respect to G if it is Markov with respect to G, but not to any proper subgraph of G.*

In other words, a distribution P has causal minimality w.r.t. G if it is Markov w.r.t. G and removing one edge from G would make P no longer Markov w.r.t. the new graph with one edge removed. This is important as the fully connected DAG is Markov w.r.t. any distribution, and often one is interested in the causally minimal graph G and not any supergraphs.

Definition 2.17 (**Causal sufficiency**). *A set of variables V is causally sufficient, if no common cause L of two variables X, Y is outside of V, i.e. for all L influencing $X, Y \in V \Rightarrow L \in V$.*

Causal sufficiency often is an important assumption, as it rules out that there is some lurking latent variable which influences two or more observed variables and thus introduces 'spurious' correlation, correlation without causal meaning. Latent variables will be discussed in Section 2.2 and their influence on causal discovery is discussed in Section 2.3.2.

The assumption in causality is that there exists some real-world, physical, data generating process which creates the probability distribution. This is mathematically formalized as a structural causal model.

Definition 2.18 (**structural causal model (SCM)**). *A structural causal model (SCM) (or structural equation model (SEM)) S is a 4-tuple (X, F, N, P_N), where*

- $N = (N_1, \dots, N_{d'})$ *is a set of background or exogenous variables whose distribution is determined by factors outside the model,*
- P_N *is a probability distribution over the domain of N,*
- $X = (X_1, \dots, X_d)$ *is a set of random variables, called observed or endogenous, which are determined by other variables in the model, i.e., variables in $X \cup N$,*
- $F = (f_1, \dots, f_d)$ *is a set of functions called structural equations, where each f_j maps from the domain of a subset of variables $N_{I'_j} \subset N$ (for $I'_j \subset \{1, \dots, d'\}$) and a subset of variables $X_{I_j} \subset X \setminus \{X_j\}$ (i.e., $I_j \subset \{1, \dots, d\} \setminus \{j\}$) to the domain of X_j, i.e., $f_j : (x_{I_j}, n_{I'_j}) \mapsto x_j$.*

The set of functions F defines a graph $G = (V, E)$, where $(X_i, X_j) \in E$ if and only if $i \in I_j$. This graph is required to be a DAG and to have $\mathrm{pa}^G_{X_j} = X_{I_j}$.

The following proposition guarantees that for any distribution with strictly positive density, an SCM with a DAG exists.

Proposition 2.19 (**Existence of SCM for All Distributions**). *Let P_X have a strictly positive density with respect to the Lebesgue measure and be Markov with respect to a DAG G. Then there exists a structural causal model (SCM) with graph G that generates P_X.*

Proof. From [Kil22]. I construct this as a noise SCM (every N_i influences exactly one X_i) with $N_1, \dots, N_d$ independently and uniformly distributed on $(0, 1)$. Now I use the (reverse) probability integral transform to set $X_j \leftarrow f_j(\mathbf{x}_{\mathrm{pa}^G_{X_j}}, N_j)$, where

$$f_j(\mathbf{x}_{\mathrm{pa}^G_{X_j}}, n) := F^{-1}_{X_j|\mathbf{X}_{\mathrm{pa}^G_{X_j}} = \mathbf{x}_{\mathrm{pa}^G_{X_j}}}(n). \tag{2.7}$$

By construction (which only works for strictly positive densities as otherwise the conditional Cumulative Distribution Function (CDF) would either not exist or not be invertible), the observational distribution of the thereby defined SCM is P_X. □

SCMs now stand on a solid mathematical basis, allowing them to describe a multitude of systems in all kinds of application fields. Together with the definitions and results on graphs, the framework introduced is capable of modeling, visualizing and answering questions about causality.In the following sections, I will expand on this causal framework.

An important limitation is that to model a system with a DAG, one assumes that any variable influencing the system can be observed (cf. Definition 2.17). This assumption often isn't met in practice. Therefore one needs to consider the possibility and handling of latent variables, which is discussed in the next Section 2.2.

2.2. Latent variables for causal graphs

This section also roughly follows the (unpublished) lecture script [Kil22]. Until now I have only dealt with the situation when all relevant variables to the system were observed, i.e. the distribution was causally sufficient (cf. Definition 2.17). If that is not the case, a DAG might no longer be able to represent well how the observed variables relate to each other. Consider the graph in Figure 2.1.

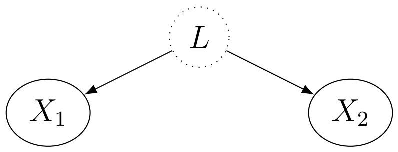

Figure 2.1.: Example for latent variable L

Imagine that only variables X_1, X_2 are observed. There is no DAG over X_1, X_2 that could represent the SCM from Figure 2.1 over these two variables only. Obviously they are correlated, as they are not d-separated by the empty set. But neither $X_1 \to X_2$ nor $X_1 \leftarrow X_2$ would describe the SCM, as X_1, X_2 do not have any direct causal influence on each other.

What is happening is that there is some latent (also called hidden, unobserved or confounding) variable that is influencing both observed variables, contradicting causal sufficiency (cf. Definition 2.17). While an SCM and associated DAG still might be the source of truth, due to some variables not being observed, the observed variables relationships and independencies can not be written in a DAG. Trying to force a DAG could lead to erroneous causal conclusions.

In addition, when latent variables are involved, it might not even be possible to find a DAG that encodes the observed conditional independencies.

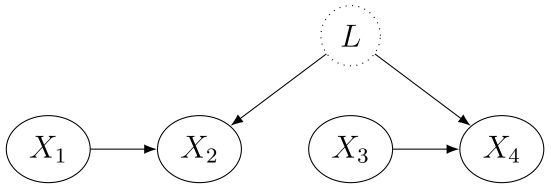

Figure 2.2.: No DAG represents observables [Kil22].

In the graph from Figure 2.2, the conditional independencies are $X_1 \perp\!\!\!\perp X_3$, $X_1 \perp\!\!\!\perp X_4$, $X_2 \perp\!\!\!\perp X_3$, $X_1 \perp\!\!\!\perp X_3 \mid X_2$, $X_1 \perp\!\!\!\perp X_3 \mid X_4$, $X_1 \perp\!\!\!\perp X_4 \mid X_3$. This means the skeleton over the observables should be as in Figure 2.3.

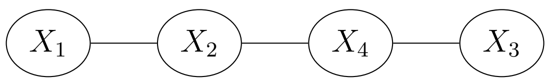

Figure 2.3.: Skeleton for graph over observables from Figure 2.2.

From the conditional independencies, neither X_2 nor X_4 can be a v-structure on this graph, because $X_1 \perp\!\!\!\perp X_3 \mid X_2$ and $X_1 \perp\!\!\!\perp X_3 \mid X_4$. So the only options left in the Markov equivalence class are shown in Figure 2.4.

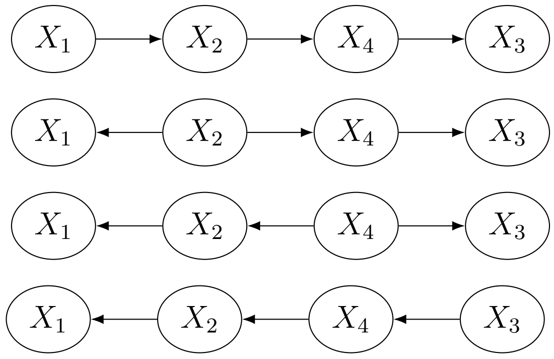

Figure 2.4.: Enumerated Markov equivalence class.

But all of these graphs would also imply $X_1 \perp\!\!\!\perp X_3 \mid X_2, X_4$, which is clearly not the case, as in the graph from Figure 2.2, X_1, X_3 are not d-separated by $\{X_2, X_4\}$.

So it is not possible to just marginalize out the latent variables. What exactly went wrong? In Figure 2.2, there is a collider at X_2 *and its adjacent node* X_4. In DAGs, it is not possible to have two adjacent nodes both being colliders on the same path. But to express the independencies clearly, one would want something like $X_1 \rightarrow X_2 \leftrightarrow X_4 \leftarrow X_3$, where the $\leftrightarrow$ represents an unobserved latent variable. The next definition formalizes this idea and builds on the previously established graph terminology in Definition 2.2.

Definition 2.20 (**Updated graph terminology**). *From now on, I will use the following terminology to also consider bidirected edges in graphs.*

(i) A directed mixed graph $G = (V, E)$ consists of a finite set of nodes V and a finite set of edges E, which are either directed ($\rightarrow$) or bidirected ($\leftrightarrow$).

(ii) A path in a graph is a sequence of distinct, adjacent edges, of any type or orientation, between distinct nodes. The first and last node on a path are the endpoints.

(iii) *A bidirected path is a path in which all edges are bidirected.*

(iv) *The notions of parent, child, ancestor, descendant, and non-descendants remain unchanged, i.e., they only refer to directed edges.*

(v) *If $X_i \leftrightarrow X_j$, I call X_i and X_j siblings and say X_i (or X_j) is a sibling of X_j (or X_i). I denote the siblings of X_j in G by $sib^G_{X_j}$.*

(vi) *The district of X_j in G for a node $X_j \in V$ is the set of nodes that are connected to X_j by a bidirected path, together with X_j itself. It is denoted by $dis^G_{X_j}$.*

(vii) *As previously, I apply the notions of parents, children, ancestors, descendants, non-descendants, siblings, and districts disjunctively to sets of nodes, i.e., $de^G_A := \bigcup_{j \in A} de^G_{X_j}$ for $A \subset V$.*

The graph that is able to encode bidirected edges is the acyclic directed mixed graph (ADMG).

Definition 2.21 (**ADMG**)**.** *An acyclic directed mixed graph (ADMG) is a directed mixed graph without directed cycles.*

The bidirected edges encode the presence of a latent variable. If the ground truth is a DAG $G = (V, E)$ over nodes $V = O \dot\cup L$ with observed variables O and latents L, one would like to project this graph down to some graph only containing the observables. This is the latent projection.

Definition 2.22 (**Latent projection**)**.** *Let $G = (V \dot\cup L, E)$ be an ADMG. The latent projection $G(V)$ of G is a directed mixed graph with nodes V, where for every pair of distinct nodes $X_i, X_j \in V$, the following hold.*

(i) *$G(V)$ contains an edge $X_i \to X_j$ if there is a directed path $X_i \to \cdots \to X_j$ on which every non-endpoint node is in L.*

(ii) *$G(V)$ contains an edge $X_i \leftrightarrow X_j$ if there exists a path between X_i and X_j such that the non-endpoints are all non-colliders in L, and such that the edge adjacent to X_i and the edge adjacent to X_j both have arrowheads at those nodes, e.g., $X_i \leftarrow \cdots \leftrightarrow X_j$.*

As a DAG is also an ADMG, Definition 2.22 also includes projection of DAGs onto ADMGs.

For the example of Figure 2.2, the latent projection would be in Figure 2.5.

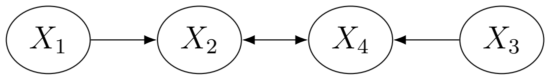

Figure 2.5.: Latent projection for graph in Figure 2.2

which now has colliders at X_2 and X_4, as previously desired.

Definition 2.23 (**Collider**). *Let $G = (V, E)$ be a directed mixed graph. A non-endpoint node X_j on a path in G is called a collider if the edges preceding and succeeding X_j on the path both have an arrow pointing to X_j. The configurations for a collider are: $\bullet \to X_j \leftarrow \bullet$, $\bullet \leftrightarrow X_j \leftrightarrow \bullet$, $\bullet \leftrightarrow X_j \leftarrow \bullet$, $\bullet \to X_j \leftrightarrow \bullet$. A non-endpoint node j on a path, which is not a collider, is termed a non-collider. The configurations for a non-collider are: $\bullet \leftarrow X_j \to \bullet$, $\bullet \leftarrow X_j \leftarrow \bullet$, $\bullet \to X_j \to \bullet$, $\bullet \leftrightarrow X_j \to \bullet$, $\bullet \leftarrow X_j \leftrightarrow \bullet$.*

Since this is a new type of graph used to express independencies, the question arises whether d-separation from DAGs carries over to ADMGs. Indeed it does by becoming m-separation, which is quite similar to d-separation.

Definition 2.24 (**M-connecting, m-separation**). *Let $G = (V, E)$ be a directed mixed graph. A path between $X_i, X_j \in V$ is called m-connecting in G given $S \subset V$ if every non-collider on the path is not in S, and every collider on the path is in S or is an ancestor of S in G. If there is no path m-connecting X_i and X_j in G given S, X_i and X_j are called m-separated given S. Sets A and B are said to be m-separated given S, if for all $X_i \in A$ and all $j \in B$, X_i and X_j are m-separated given S.*

The idea remains the same: colliders are blocked by default, while a non-collider would have to be blocked to achieve m-separation between variables. In a DAG, d-separation and m-separation are the same. The only difference is that due to the addition of bidirected edges, more possibilities for collider and non-collider structures arise in ADMGs.

The definitions and theorems of this section are important for the introduction of C-ADMGs (Cluster-ADMGs), which are ADMGs over clusters of variables, in Section 3.1. The next section introduces causal discovery.

2.3. Causal discovery

Armed with the notations and definitions of the previous sections I am ready to step closer to the articulated goals of this thesis, which are in the domain of causal discovery. Until now, the notation and theorems were mostly focused on the case when the DAG is known and how to deal with that. In problems arising in real world applications, the DAG is often not known a priori and one would like to find it. This leads to the task of causal discovery, where one tries to estimate the ground truth DAG (or something close to it) from data. In the following, I will briefly present the main types of algorithms used in causal discovery and then dive deeper into the discovery methods of choice for this thesis, the constraint based algorithms. Other reviews of causal discovery algorithms have been done, for example [HMM18], [GZS19], [ZOS22] or [HHG23].

Apart from constraint based discovery, there are several other types of discovery algorithms. I will briefly discuss discovery via structural assumptions, score-based search and score based continuous optimization, to put the constraint based methods into a richer context. More exist, e.g. hybdrid methods, which combine ideas from constraint based and score based methods or BACKSHIFT [Rot+15], which exploits invariance properties. Methods for time series data exist, a survey can be found in [ADG22]. Methods can also include interventional (experimental) data like [HB12]. Discussing all of these in detail goes beyond the scope of this thesis and I refer to the respective papers and surveys cited for further information.

Constraint based discovery starts from a fully connected graph and finds the ground truth by removing edges when a conditional independence is found. For this to be successful, the assumption of faithfulness is crucial, i.e. when $X \perp\!\!\!\perp Y \mid Z$ is observed in the data, if the distribution is faithful w.r.t. the ground truth DAG G, that would imply $X \perp\!\!\!\perp_G Y \mid Z$ in the graph. With that information, the edge $X - Y$ can be safely removed, as adjacent vertices have the property that no set can d-separate them. Because a d-separating set was found, X and Y cannot be adjacent.

The algorithm then iteratively prunes the graph by performing (conditional) independence tests and removing edges accordingly. Since the set of possible d-separating sets is very large, the algorithms need to employ some clever tactics to traverse this space efficiently. The details of the PC and FCI algorithms will be discussed in Sections 2.3.1 and 2.3.3.

Structural assumptions put certain assumptions on the SCM and make discovery possible that way. One way of doing this is to assume the ground truth to be an additive noise model (ANM). Then the equations from the SCM take the following form:

$$\begin{aligned} X &= N_y \\ Y &= X + N_x \end{aligned} \tag{2.8}$$

N_x, N_y introduce the noise. If the noise is non-Gaussian, performing a linear regression and looking at the residuals will reveal the causal direction. Consider the example in Figure 2.6 from [PJS17] Chapter 4. Let N_x, N_y be uniform $U[-0.5, 0.5]$ random variables with X, Y relating to each other as in Equation 2.8. The red line on the left in Figure 2.6 is a linear regression of Y on X, i.e. assuming the formula $Y = \alpha X + N_x$ (which can model the true data generating process). In this case the residuals are independent. When regressing X on Y on the other hand (assuming the formula $X = \alpha Y + N_y$, not the true data generating process), which gives the blue line on the right, the residuals are not independent. This reveals the causal direction $X \to Y$.

The reason this works only in the non-gaussian case is that for gaussian noise, the residuals will spread in a two dimensional bell-curve around the origin. This means the residuals will be independent from both the causal and anti-causal regression point of view and causal structure identification is impossible. This idea is used in the LiNGAM algorithm from [Shi14], where LiNGAM stands for linear non-gaussian additive noise.

Score based search uses a criterion, the score, to evaluate the fit of a graph G to the data D. The score usually is a penalized likelihood score, like the Bayesian information

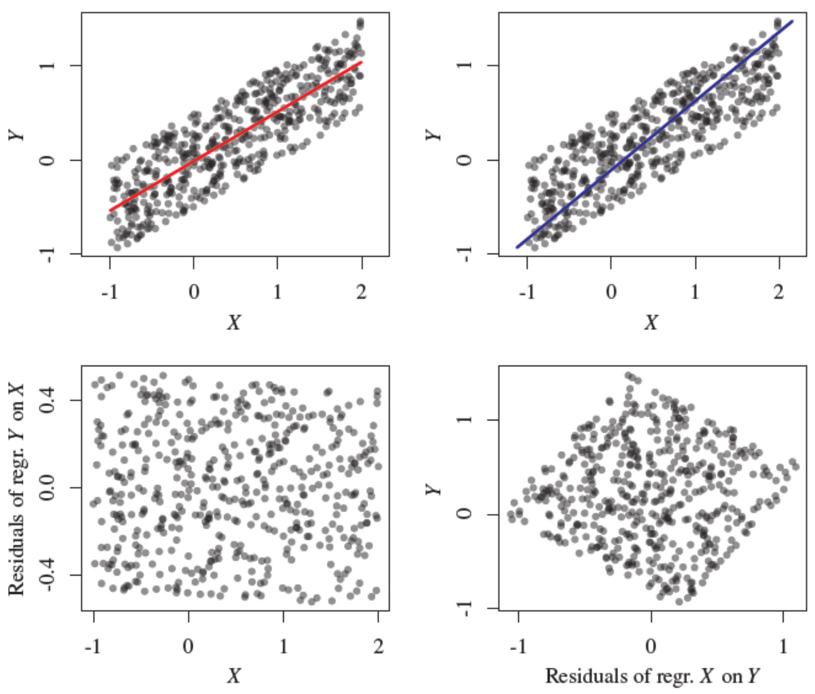

Figure 2.6.: Causal discovery with non-gaussian ANM. Left: linear regression (red line) of Y on X in causal direction with independent residuals. Right: linear regression (blue line) of X on Y in anti-causal direction X on Y with dependent residuals. Figure from [PJS17] Chapter 4.

criterion (BIC) [Chi02]:

$$BIC(G, D) = \log(p(D \mid \hat{\theta_G}, G)) - \frac{d}{2}\log(n), \tag{2.9}$$

where $\hat{\theta}_G$ denotes the parameters for the log-likelihood, G is the graph, d is the dimension of the graphical model and n is the amount of data points.

This leads to the optimization problem in Equation 2.10, with $S(W)$ the score function of the weighted adjacency matrix W.

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times d}} \quad & S(W) \\ \text{subject to} \quad & G(W) \in \text{DAGs}. \end{aligned} \tag{2.10}$$

The constraint $G(W) \in$ DAGs (the graph induced by the adjacency matrix being a DAG) poses a big challenge: the space of DAGs is superexponentially big in the number of vertices. Why? It is easy to see that a graph with d nodes can contain at maximum $\frac{d(d-1)}{2}$ edges (connect the first node with $d-1$ other nodes, the second node can at most be connected to $d-2$ new nodes, etc., apply Gauß's formula for sums of integers $\sum_{i=1}^{d-1} i = \frac{d(d-1)}{2}$). This means in $\frac{d(d-1)}{2}$) locations, an edge could be $\rightarrow$, $\leftarrow$ or absent ($\nrightarrow$).

These are three distinct possibilities, so the number of possible directed graphs is $3^{\frac{d(d-1)}{2}}$. Of course some graphs contain cycles, these should be eliminated to find the number of DAGs. So $3^{\frac{d(d-1)}{2}}$ is an upper bound. And $2^{\frac{d(d-1)}{2}}$ is a lower bound: if one orders all nodes to $\{X_1, ..., X_d\}$ and only allows edges to be either $X_i \to X_j$ for $i < j$ or absent, those graphs are guaranteed to be acyclic (due to the forced topological ordering). Since they have two possibilities of edges, $\to$ or absent, their number is $2^{\frac{d(d-1)}{2}}$. To conclude, $2^{\frac{d(d-1)}{2}} < \#DAGs(d) < 3^{\frac{d(d-1)}{2}}$, which is superexponential (due to the square in the exponent) in the number of nodes d.

The most famous score based search algorithm, greedy equivalence search (GES, [Chi02]), tackles this search problem with a greedy approach. It starts with an empty graph and scores it. Then it looks at all possible single edge additions and chooses the one that increases the score the most. This is done as long as it is possible to increase the score, which is called the forward phase. When the score can no longer be increased by adding edges, the backward phase begins. It looks at single edge deletions and chooses the one which leads to the biggest score increase. When the score can no longer be improved by deleting edges, the graph (strictly speaking, the graph representing a Markov equivalence class) is returned. This procedure tries to optimize by only looking at the next step best increase, it is called greedy. Surprisingly, despite the greedy approach, [Chi02] showed that GES is consistent (under some assumptions) and [NMR17] show high-dimensional consistency of GES. There also exists GIES, greedy interventional equivalence search from [HB12] which allows inclusion of interventional data.

Continuous score based methods like DAGs with NO TEARS [Zhe+18] transform the discrete optimization problem from Equation 2.10 to a continuous one by reformulating the acyclicity constraint from $G(W) \in DAGs$ into a smooth function $h(W)$ which is zero iff $G(W)$ is acyclic. Recall W is the weighted adjacency matrix of a DAG. If $X_i \to X_j$ in the DAG G, $W_{i,j} \neq 0$, otherwise $W_{i,j} = 0$. h is powered by the trace of the matrix exponential, $h(W) = tr(e^{W \circ W})$. $W \circ W$ is to make the entries in the matrix all ≥ 0. The trace can 'count' the number of cycles, and if $tr(e^{W \circ W}) = d$, where d is the dimension of W, then $G(W)$ is acyclic. If $tr(e^{W \circ W}) > d$, there are some cycles, so h measures the 'DAG-ness' of the graph $G(W)$. This transforms the optimization problem:

$$\begin{array}{ll} \text{Minimize} & S(W) \\ \text{subject to} & G(W) \in \text{DAGs} \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \text{Minimize} & S(W) \\ \text{subject to} & h(W) = 0, \end{array} \tag{2.11}$$

where now both functions S and h are differentiable and hence allow for continuous optimization algorithms. These algorithms have been shown to perform well, albeit [RSW21] caution that good performance may be partly due to var-sortability of simulated DAGs.

2.3.1. PC algorithm

The PC algorithm [Spi+00] is one of the first and most well-known constraint based discovery algorithms. It relies on the global Markov property, faithfulness and causal sufficiency. This means the Bayesian network has a one to one correspondence between conditional independence and d-separation and its graph is a DAG. The algorithm starts from a fully connected undirected graph. Using faithfulness, i.e. $X \perp\!\!\!\perp Y \mid Z \Rightarrow X \perp\!\!\!\perp_G Y \mid Z$, if a conditional independence is found, the variables X, Y can be d-separated in the graph. Hence they cannot be adjacent, and their edge can be removed.

The theoretic version of PC assumes access to an independence oracle. In practice, this does not exist, so instead one relies on conditional independence tests like Fisher-z or kernel conditional independence tests (KCI). Conditional independence testing is challenging due to the curse of dimensionality, particularly for continuous variables [Zha+12]. As this does not directly concern this thesis, I will not discuss this problem here.

A second issue in PC is that there are a lot of conditional independence tests $X \overset{?}{\perp\!\!\!\perp} Y \mid Z$ that could be performed - all $Z \subset V \setminus \{X, Y\}$ would be possible. A key question is how to search this set in a sensible and fast way.

Lastly, Theorem 2.14 states that from conditional independence alone, only the skeleton and v-structures can be recovered. So instead of G, only $M(G)$, the Markov equivalence class, is learnable from data. This equivalence class can be expressed as a completed partially directed acyclic graph (CPDAG), which was already mentioned in Definition 2.13).

Definition 2.25 (CPDAG (completed partially directed graph), [AMP97]). *The completed partially directed graph of a DAG G, denoted by G^*, is a graph with the same skeleton as G and undirected edges. A directed edge occurs if and only if that directed edge is present in all DAGs of the Markov equivalence class of G.*

As an example, for the graph $X_1 \to X_2$, the CPDAG would be $X_1 - X_2$, as the Markov equivalence class consists of $X_1 \to X_2$ and $X_1 \leftarrow X_2$. For the graph $X_1 \to X_2 \leftarrow X_3$, the CPDAG would be the same graph, $X_1 \to X_2 \leftarrow X_3$, because the v-structures orient all edges.

In addition to orienting some edges via v-structures, additional orientation rules can be used due to information about v-structures and the acyclicity constraint. These so called Meek's rules are visualized in Figure 2.7.

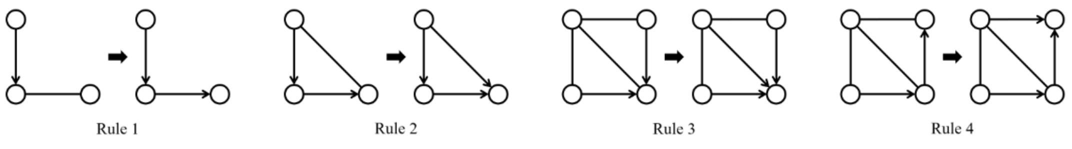

Figure 2.7.: Meek's orientation rules [Mee13] (Figure from [Fan+22]).

The first rule has to orient the lower arrow to the right. If the arrow was to the left, it would have been part a v-structure, which would have been detected. Since that isn't the case, the arrow must be oriented to the right, not forming a v-structure. The second rule is due to acyclicity. In the third rule, as there is no v-structure in the top left, either the left edge is pointing down or the top edge is pointing right. Either way, rule two would find application for the vertical edge, so it has to be oriented towards the bottom right. The fourth rule is due to acyclicity and v-structures. The top left was not identified as a v-structure. If the top edge was oriented to the left, either there would be a cycle or a v-structure. So it has to be oriented towards the right.

It is now possible to state the PC algorithm.

Algorithm 1 PC algorithm

```
Require: Joint distribution $P_X$ of $d$ variables, independence oracle
 1: $V \leftarrow \{X_1, \dots, X_d\}$, $E \leftarrow \{(X_i, X_j) \in V^2 | i \neq j\}$   ▷ form fully connected undir. graph
 2: for $k = 0, \dots, d-2$ do                                        ▷ find skeleton
 3:     for each adjacent pair $X_i, X_j \in V$ do
 4:         for all $S \subset V \setminus \{X_i, X_j\}$ with $|S| = k$ (and $S - X_i$ or $S - X_j$) do
 5:             if $X_i \perp\!\!\!\perp X_j | S$ then
 6:                 $E \leftarrow E \setminus \{(X_i, X_j), (X_j, X_i)\}$            ▷ remove $X_i - X_j$
 7:                 $S_{i,j} \leftarrow S$
 8: for each triple $X_i, X_j, X_k \in V$ with
    $X_i - X_j - X_k$ and $X_i \not- X_k$ do                              ▷ find v-structures
 9:     if $X_j \notin S_{i,k}$ then
10:         $E \leftarrow E \setminus \{(X_j, X_i), (X_j, X_k)\}$      ▷ orient the edges as $X_i \rightarrow X_j \leftarrow X_k$
11: Apply Meek's edge orientation rules from Figure 2.7
12: successively apply Meek's orientation rules from Figure 2.7
13: return CPDAG $G = (V, E)$
```

The algorithm works in three phases. In the first phase, the skeleton is recovered by removing edges between nodes for whom a conditional independence can be found. The second phase orients all the v-structures and the third phase performs additional edge orientations according to Meek's rules.

The number of potential separating sets is vast and in practice one tries to eliminate edges as soon as possible. For this, the algorithm performs the 'easy' conditional independence tests, meaning the ones with conditioning set of low cardinality, first. The algorithm increases the size of $|S|$ incrementally - that is the 'depth' k. First all separating sets of cardinality zero are considered (that means independence tests without conditioning). After all of them have been exhausted, the algorithm moves on to $|S| = 1$, and so forth. With that strategy, the easier independence tests are performed first.

More speedup is needed still, so a second strategy comes into play. Testing for all $S \subset V \setminus \{X_i, X_j\}, |S| = k$ is still a lot. Right now, the superset that must include a separating set is $V \setminus \{X_i, X_j\}$, so all of its subsets of cardinality $|k|$ are being considered. It would be desirable if one could reduce the supersets size and fortunately, this is possible. Consider the minimal neighbor separator from [GCL23].

Definition 2.26 (**mns (minimal neighbor separator)** [GCL23])**.** *For a DAG $G = (V, E)$ and node X and $A \notin nb_X^+$ ($nb_X^+ := nb_X \cup \{X\}$), the minimal neighbor separator $mns_X(A) \subset nb_X$ is the unique set of nodes such that*

(i) (d-separation) $A \perp\!\!\!\perp_G X \mid mns_X(A)$

(ii) (minimality) for any $S \subset mns_X(A) : A \not\!\perp\!\!\!\perp_G X \mid S$

hold.

This allows for the following propositions:

Proposition 2.27 (**Restricting separating set via mns** [GCL23])**.** *For any node $Y \notin de_X \cup nb_X^+$, $mns_X(Y)$ exists and $mns_X(Y) \subset pa_X$.*

Proof. See [GCL23]. □

Proposition 2.28 (**Uniqueness of mns** [GCL23])**.** *For every node Y such that $mns_X(Y)$ exists, it is unique.*

Proof. See [GCL23]. □

The $mns_X(Y)$ is the smallest set blocking all paths between X and Y. To come back to the speed-up strategy, consider that for any X, Y in a DAG which are not adjacent ($Y \notin nb_X^+$ and $X \notin nb_Y^+$), either $Y \notin de_X$ or $X \notin de_Y$, so by Proposition 2.27 either $mns_X(Y) \subset pa_X$ or $mns_Y(X) \subset pa_Y$. So when searching for d-separating sets between X and Y (and therefore sets making X and Y conditionally independent), it suffices to look at their parents as supersets. Since a priori the graph contains undirected edges and it is not clear which neighbors of X or Y are parents, one then looks in nb_X^G and nb_Y^G for potential separating sets (G is the current graph of the PC algorithm, where some edges were already removed by previous conditional independence tests). These strategies have the great benefit of reducing the number of needed conditional independence tests the more edges can be removed, so the algorithm automatically works faster for sparse graphs, where a lot of edges can be quickly removed.

In practice, the algorithm is usually altered to eliminate the order dependence of PC. PC is order dependent due to the edges being removed immediately and therefore altering the neighbor sets for future conditional independence tests. [CM+14] therefore propose to postpone edge deletions to the end of each depth phase. All edges that could be deleted are saved temporarily, and only before the depth is increased by one are the edges deleted from the current graph. This means the results of each depth phase will not differ depending on the variable ordering, as the graph is not changed during a depth phase. Hence the algorithm is guaranteed to produce the same output, even when the variables are ordered differently.

More extensions and modifications to the PC algorithm have been proposed, like multi-core processing and parallelizing strategies ([Le+16] , [Zha+21a]). [KB07] prove that the PC algorithm is uniformly consistent under a minimal sparseness assumption. [HD13] introduce the rankPC, which deals with the problem of conditional independence testing by replacing the standard independence test based on pearson correlations with rank based measures of correlation.

2.3.2. Constraint based causal discovery with latent and selection variables

The PC algorithm from the previous section has a serious limitation: It assumes that all variables of interest are observed, i.e. Definition 2.17 causal sufficiency holds and there are no latent variables. This assumption is often not met in practice, as already discussed in Section 2.2. That can make the algorithm produce a false output from which erroneous conclusions would be drawn.

For causal discovery the problem from unobserved variables is twofold: latent confounders and selection variables. I will show how they cause issues with causal discovery one by one.

Recall that a latent confounder is an unobserved variable that influences several other observed variables. The example in Figure 2.1 showed that a DAG can not accurately represent the relationships between variables when latent confounders are present. Furthermore the example in Figure 2.2 showed that even if the distribution is faithful w.r.t. the true DAG (including latents), there may be no DAG over the observables that is faithful w.r.t. the marginal distribution over the observed variables.

Another problem comes from selection variables. These are variables that the dataset is involuntarily conditioned on. While latent confounders influence the observed variables from above, selection variables work from the opposite side, from below.

As an example consider the case of a medical study, attributed to Chris Meek in [Zha08b], originally from [Ric98]:

The graph in Figure 2.8 represents a randomized trial of an ineffective drug with unpleasant side-effects. Patients are randomly assigned to the treatment or control group (A). Those in the treatment group suffer unpleasant side-effects (Ef), the severity of which is influenced by the patient's general level of health (H), with sicker patients suffering worse side-effects. Those patients who suffer sufficiently severe side-effects are likely to drop out of the study. The selection variable (Sel) records whether or not a patient remains in the study, thus for all those remaining in the study $Sel = StayIn$. Since unhealthy patients who are taking the drug are more likely to drop out, those patients in the treatment group who remain in the study tend to be healthier than those in the

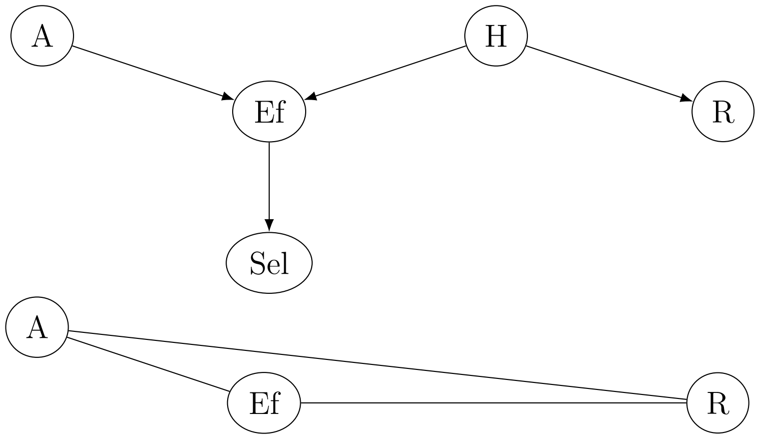

Figure 2.8.: Above: graph with latent and selection variable, below: resulting CPDAG

control group. Finally health status (H) influences how rapidly the patient recovers (R) ([Ric98], p. 234).

In the dataset drawn from a distribution faithful to the graph in Figure 2.8 and marginalized to $\{A, Ef, R\}$, the selection variable Sel will be conditioned on. This means the collider at Ef "opens" and $A \not\perp\!\!\!\perp R$ will be observed. The resulting CPDAG therefore will be as in the second part of Figure 2.8. This graph does not represent the conditional independence relations well and leads to the erroneous conclusion that treatment A is correlated with recovery R, and since A happens before R, one could falsely postulate $A \rightarrow R$.

To include latent variables, it seems straightforward to look at the ADMGs from Section 2.2 and try to estimate those, or a CPDAG analogue for ADMGs, maybe a completed partially directed acyclic mixed graph (CPDAMG). Unfortunately, it is not possible to learn ADMGs or something like a CPDAMG from conditional independence tests (CI tests). Why this is the case becomes clear from the following example. Consider the graph in Figure 2.9.

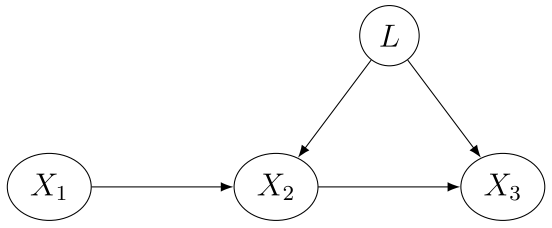

Figure 2.9.: Example graph for causal discovery with latent variables.

The latent projection (cf. Definition 2.22) would be the graph in Figure 2.10.

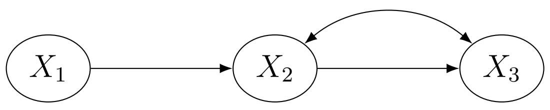

Figure 2.10.: Latent projection for causal discovery with latent variables.

But what would be the output of the PC algorithm? Remember that it starts from a fully connected graph, and first finds the skeleton. Of course edges $X_1 - X_2$ and $X_2 - X_3$ would remain. What about $X_1 - X_3$? It is $X_1 \not\perp\!\!\!\perp X_3$ due to the active path $X_1 \rightarrow X_2 \rightarrow X_3$. But it is also $X_1 \not\perp\!\!\!\perp X_3 \mid X_2$ due to the path $X_1 \rightarrow X_2 \leftarrow L \rightarrow X_3$ being activated by conditioning on collider X_2. It would be possible to block that graph again by also conditioning on L, i.e. $X_1 \perp\!\!\!\perp X_3 \mid X_2, L$, but that CI test cannot be done as L is latent, so no data on it was collected. Thus the output of PC would be the fully connected undirected graph as CPDAG, with $X_1 - X_3$ being present, even though X_1 and X_3 are neither adjacent in the ground truth DAG of Figure 2.9 nor adjacent in the ADMG of Figure 2.10.

The following went wrong: X_1 and X_3 are not d-separable purely from the observed variables, one would have to include L. The above example shows that neither a DAG, CPDAG, ADMG or CPDAMG can be the target output for an algorithm that takes latents into consideration.

To deal with these problems ancestral graphs are introduced. The algorithm that is able to learn these graphs from data is the FCI algorithm ([Spi+00], [SMR13], [Zha08b]), which stands for fast causal inference, although the 'fast' in the name is considered optimistic.

Ancestral graphs, like ADMGs, are mixed graphs and include bidirected edges, but they can also include undirected edges (these are for selection variables). My explanation follows [Zha08b].

Definition 2.29 (**Graph terminology for ancestral graphs**). *In addition to previously introduced graph terminology, the following definitions will be needed for ancestral graphs:*

(i) The two ends of every edge are called marks or orientations. A mark can either be an arrowhead ($<$ or $>$), a tail ($-$). A directed edge contains one of each, a bidirected edge has two arrowheads and an undirected edge has two tails. An edge is into (or out of) a node if the edge mark at that vertex is an arrowhead (or a tail).

(ii) A circle $\circ$ is used to indicate uncertainty over whether the mark is an arrowhead or a tail. A star $*$ will be used in orientation rules as a wildcard for either an arrowhead, a tail or a circle.

(iii) An almost directed cycle between X and Y occurs in G when $X \leftrightarrow Y$ is in G and $X \in an_Y^G$.

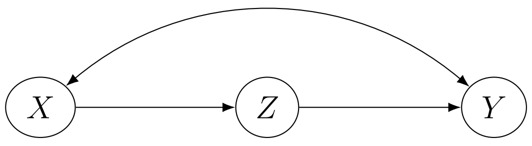

Figure 2.11.: Example of an almost directed cycle $X \to Z \to Y \leftrightarrow X$.

Definition 2.30 (**Ancestral graph**). *A mixed graph G is ancestral if the following three conditions hold:*

(i) there is no directed cycle,

(ii) there is no almost directed cycle,

(iii) for any undirected edge $X_1 - X_2$, X_1 and X_2 have no parents or siblings.

From the definition it follows that ancestral graphs contain at most one edge between nodes. If $X \to Y$ and $X \leftrightarrow Y$ were in an ADMG, $X \to Y$ would be left in a corresponding ancestral graph. DAGs are a special case of ancestral graphs, as Definition 2.30 (i) indicates. Together with Definition 2.30 (ii), acyclicity gives arrowheads in ancestral graphs the meaning of non-ancestorship. Consider the latent projection of the graph in Figure 2.11. In that graph, $X_1 \leftrightarrow X_3$, which would be forbidden in ancestral graphs. The edge would be $X_1 \to X_3$. This means if an edge between X_1, X_3 is into X_3, i.e. $X_1 {*\!\!\to} X_3$, X_3 is not an ancestor of X_1. The no almost directed cycle rule enforces this. To see how this would be a contradiction if there was an almost directed cycle, consider the latent projection of Figure 2.11. The edge $X_1 \leftrightarrow X_3$ is into X_1, but X_1 is an ancestor of X_3. Therefore, the arrow head at X_1 does not imply 'X_1 is not an ancestor of X_3'.

Conveniently, the notion of m-separation from Definition 2.24 for ADMGs carries over to ancestral graphs. Unfortunately, it is possible for two vertices to not be adjacent in an ancestral graph, but also not have any set able to m-separate them. The graph on the left hand side in Figure 2.12 shows how this can happen. X_3 and X_4 are not adjacent, yet it is not possible to m-separate X_3 from X_4. Whatever combination one takes out of $\{X_1, X_2\}$ to block, either a collider path is opened up or a non-collider path stays unblocked.

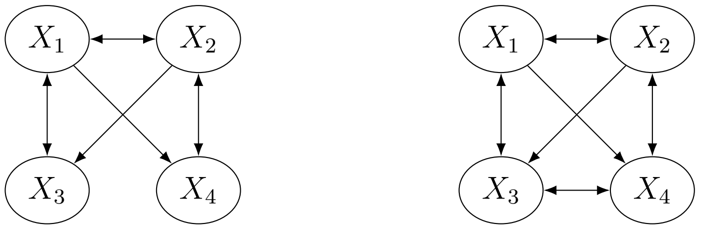

Figure 2.12.: Left: ancestral graph that is not maximal. Right: maximal ancestral graph (example from [Zha08b])

This is inconvenient when trying to learn graphs via conditional independence/ m-separation, as one cannot hope to delete an edge between non-adjacent vertices when no possible m-separating set exists for them. For this reason, maximal ancestral graphs, in whom non-adjacency does imply existence of an m-separating set, are needed.

Definition 2.31 (**MAG (maximal ancestral graph)**. *An ancestral graph is called maximal if for any two non-adjacent vertices, there is a set of vertices that m-separates them.*

As [Zha08b] explains, DAGs are all maximal. Maximality goes hand in hand with inducing paths. Their definition is quite complicated, but the basic idea is like this: When are two non-adjacent vertices not m-separable in an ancestral graph? When there exists an inducing path between them.

Definition 2.32 (**Inducing path** [Zha08b])**.** *In an ancestral graph, let X, Y be any two vertices and L, S be disjoint sets of vertices not containing X, Y. L describes the latent variables and S describes the selection variables. A path π between X and Y is called an inducing path relative to (L, S) if every non-endpoint vertex on π is either in L or a collider, and every collider on π is an ancestor of either X, Y, or a member of S.*
When $L = S = \emptyset$, π is called a primitive inducing path between X and Y.

The motivation behind this complicated definition can be explained as follows. The idea of inducing paths is that they are unblockable. When is that the case? Consider the example from Figure 2.9, shown again below in Figure 2.13.

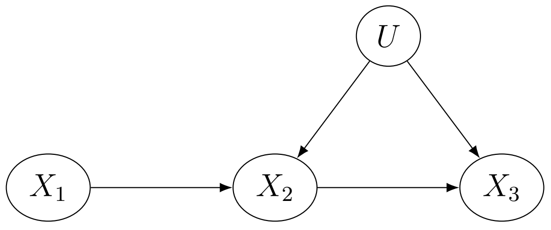

Figure 2.13.: Graph with inducing path from $X_1 \rightarrow X_2 \leftarrow U \rightarrow X_3$

In that graph, it is impossible to m-separate X_1 from X_3 with only the observed variable $\{X_3\}$. This is because the path $X_1 \rightarrow X_2 \leftarrow U \rightarrow X_3$ is inducing. The non-endpoints are X_2, U. $U \in L$ is in the latent variables and X_2 is a collider *and* that collider is an ancestor of either X_1 or X_3 - in this case X_3 (one could have also made it inducing by putting X_2 in the selection variables). So it is impossible to m-separate X_1 from X_3 using only the observed variables X_1, X_2, X_3: including X_2 in the separating set opens the path $X_1 \rightarrow X_2 \leftarrow U \rightarrow X_3$ and excluding X_2 leaves the path $X_1 \rightarrow X_2 \rightarrow X_3$ open.

This leads to a maximality criterion for ancestral graphs via inducing paths.

Proposition 2.33 (**Maximality criterion for ancestral graphs** [Zha08b])**.** *An ancestral graph is maximal if and only if there is no primitive inducing path between any* *two non-adjacent vertices in the graph.*

Proof. See [Zha08b]. □

For example, the graph on the left in Figure 2.12 contains the primitive inducing path $X_3 \leftrightarrow X_1 \leftrightarrow X_2 \leftrightarrow X_4$, because $X_1 \rightarrow X_4$ and $X_2 \rightarrow X_3$. In a DAG a primitive inducing path is just one edge, either $X \rightarrow Y$ or $X \leftarrow Y$.

[RS02] Theorem 4.2 establishes that X and Y are not m-separable if and only if there exists an inducing path relative to (L, S) between them. Thus in MAGs, existence of an m-separating set corresponds to non-adjacency and vice versa.

MAGs serve the purpose of modelling DAGs without needing to take L and S into consideration, they are like a projection of the conditional independencies down to the observable variables. The question is then how to construct a MAG G_M from a DAG G with $V = O \cup L \cup S$, so that one can use the MAG as the graph of reference regarding conditional independencies.

Algorithm 2 DAG to MAG [Zha08b]

```
Require: DAG $G$ over $V = O \cup L \cup S$
1: for each pair of variables $X, Y \in O$, $X$ and $Y$ are adjacent in $G_M$ if and only if there
   is an inducing path relative to $(L, S)$ between them in $G$
2: for each pair of adjacent vertices $X, Y$ in $G_M$ do
3:     orient edge $X \rightarrow Y$ in $G_M$ if $X \in an^G_{\{Y\}\cup S}$ and $Y \notin an^G_{\{X\}\cup S}$
4:     orient edge $X \leftarrow Y$ in $G_M$ if $Y \in an^G_{\{X\}\cup S}$ and $X \notin an^G_{\{Y\}\cup S}$
5:     orient edge $X \leftrightarrow Y$ in $G_M$ if $X \notin an^G_{\{Y\}\cup S}$ and $Y \notin an^G_{\{X\}\cup S}$
6:     orient edge $X - Y$ in $G_M$ if $X \in an^G_{\{Y\}\cup S}$ and $Y \in an^G_{\{X\}\cup S}$
7: return MAG $G_M$ over $O$
```

The MAG G_M probabilistically represents G, which follows from [RS02] Theorem 4.18. In addition, according to [Zha08b], G_M also causally represents G by retaining ancestral relationships. Specifically,

- $X \rightarrow Y$ means that X is a cause of Y or of some selection variable, but Y is not a cause of X or of any selection variable,
- $X \leftrightarrow Y$ means that X is not a cause of Y or of any selection variable, and Y is not a cause of X or of any selection variable,
- $X - Y$ means that X is a cause of Y or of some selection variable, and Y is a cause of X or of some selection variable.

In essence, arrowheads mean negative causal information, information about non-cause, while tails mean positive causal information about cause [Zha08b]. If selection bias is allowed, the positive causal information is less informative. If no selection bias is assumed, $X \to Y$ means X is a cause of Y [Zha08b].

As with DAGs, two different MAGs, while carrying different causal information could share the exact same m-separation structure [Zha08b].

Definition 2.34 (**Markov equivalence in MAGs** [Zha08b])**.** *Two MAGs G_M, H_M with the same vertices are Markov equivalent if for any three disjoint sets of vertices X, Y, Z, X and Y are m-separated by Z in G_M if and only if X and Y are m-separated by Z in H_M.*

As in DAGs, MAGs can also contain v-structures, also called unshielded colliders.

Definition 2.35 (**Unshielded path** [Zha08b])**.** *In a MAG, a path consisting of a triple of vertices (X, Y, Z) is said to be unshielded if X and Z are not adjacent. The triple is called an unshielded collider if both the edge between X and Y and the edge between Y and Z are into Y.*

Theorem 2.14 shows that two DAGs are Markov equivalent if and only if they posess the same adjacencies and v-structures. That is also necessary for MAGs, but not sufficient. The reason for this is the existence of discriminating paths.

Definition 2.36 (**Discriminating path** [Zha08b])**.** *In a MAG, a path between X and Y, $\pi = (X, ..., W, S, Y)$ is a discriminating path for S if*

(i) π includes at least three edges,

(ii) S is a non-endpoint vertex on π and is adjacent to Y on π,

(iii) X is not adjacent to Y and every vertex between X and S is a collider on π and is a parent of Y.

Proposition 2.37 (**Markov equivalence criterion for MAGs** [Zha08b])**.** *Two MAGs over the same set of vertices are Markov equivalent if and only if*

(i) they have the same adjacencies,

(ii) they have the same unshielded colliders,

(iii) if a path π is a discriminating path for a vertex S in both graphs, then S is a collider on the path in one graph if and only if it is a collider on the path in the other.

Proof. See [Zha08b]. □

Why is it necessary to add discriminating paths? As an example one can examine why the two graphs in Figure 2.14 do not exhibit the same m-separation structure, even though they have the same adjacencies and unshielded colliders (none in this case).

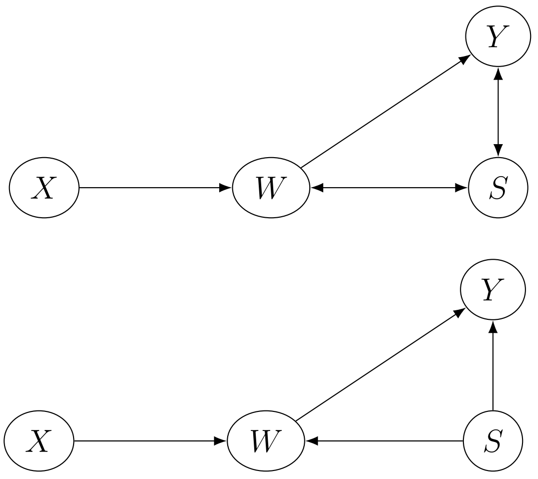

Figure 2.14.: Example for a discriminating path between X and Y for S

To start, both graphs are indeed MAGs, as they contain no inducing paths. In the first graph, $X \rightarrow W \leftrightarrow S \leftrightarrow Y$ almost looks like an inducing path, but the collider S is not an ancestor of X or Y. The same is true for $X \rightarrow W \leftarrow S \rightarrow Y$ in the second graph. Therefore in both graphs it is possible to m-separate X and Y. Note that they both contain the discriminating path (X, W, S, Y) for S.

In the first graph, it is $X \perp\!\!\!\perp_G Y \mid W$, as that blocks the directed path $X \rightarrow W \rightarrow Y$, and while it opens the collider path $X \rightarrow W \leftrightarrow S \leftrightarrow Y$, that path is still blocked by not conditioning on S, as it is a collider. In the second graph, the situation is different, here $X \not\perp\!\!\!\perp_G Y \mid W$, as the path $X \rightarrow W \leftarrow S \rightarrow Y$ is not blocked. In the second graph it is $X \perp\!\!\!\perp_G Y \mid W, S$. This m-separation does not hold in the upper graph, as that would open the aforementioned collider path $X \rightarrow W \leftrightarrow S \leftrightarrow Y$.

The example explains why discriminating paths have to be included in Proposition 2.37. Without discriminating paths, two graphs with different m-separation structures could be put into the same Markov equivalence class. That has to be avoided, as the Markov equivalence class is meant to represent the exact same conditional independence statements.

The statements about Markov equivalence mean that again, as for DAGs, only the Markov equivalence class of a MAG is recoverable from conditional independence testing. One would like to characterize this Markov equivalence class with a graph, analogous to the CPDAG for DAGs. This is the PAG (partial ancestral graph), which will also be the output type of the FCI algorithm. It uses circle $\circ$ edge marks to indicate uncertainty on whether an edge mark is an arrow head or a tail (for DAGs undirected edges were used to indicate uncertainty about its direction, but they serve a different purpose for selection variables now).

Definition 2.38 (**Partial ancestral graph** [Zha08b]). *Let $M(G)$ be the Markov equivalence class of a MAG G. A partial ancestral graph (PAG) for $M(G)$ is a graph G_P*

with possibly three kind of edge marks (and hence six kinds of edges: $-$, $\rightarrow$, $\leftrightarrow$, $\circ\!-$, $\circ\!-\!\circ$, $\circ\!\!\rightarrow$), such that

(i) G_P has the same adjacencies as G (and any member of $M(G)$ and

(ii) every non-circle mark in G_P is an invariant mark in $M(G)$.

If furthermore every circle in G_P corresponds to a variant mark in $M(G)$, G_P is called the maximally informative PAG for $M(G)$.

2.3.3. FCI algorithm

With the preliminaries on ancestral graphs from Section 2.3.2 completed, it is now possible to state the FCI algorithm. It works similarly to the PC algorithm, but takes potential bidirected edges (and therefore latent confounders) into consideration.

The FCI algorithm was originally described in [Spi+00] Section 6.7 with the target output being a partially oriented inducing path graph (POIPG), which can also be interpreted as a PAG, as [Zha08a] Appendix A explains in detail. In short, ancestral graphs are a subclass of inducing path graphs. After re-interpretation of its output as described before, where selection variables put an 'or' in the arrow interpretations, FCI also takes selection variables into account.

In this section, I present the FCI algorithm from [Col+12a]'s supplement [Col+12b]. FCI assumes that the distribution over all variables $P_V, V = O \cup L \cup S$ is faithful w.r.t. the ground truth DAG over V.

In the PC algorithm, it sufficed to search for separating sets between X and Y in their adjacent nodes adj_X, adj_Y. In a MAG, that is no longer possible. Consider the graph in Figure 2.15. It is a MAG, as it is ancestral and there are no non-adjacent nodes that cannot be m-separated. But that does not mean all nodes can be m-separated by some subset of their adjacencies. $adj_{X_1} = \{X_2, X_4\} = adj_{X_5}$, but $X_1 \not\perp\!\!\!\perp X_5 \mid S$ for any $S \subset \{X_2, X_4\}$. It is not possible to condition on X_2 or X_5, as that would open collider paths, but not conditioning on them would keep non-collider paths open. The path $(X_1, X_2, X_3, X_4, X_5)$ seems similar to an inducing path, as colliders on the path are also ancestors of endpoints. The difference is that the non-collider X_3 is not in the latent variables. If X_3 was also latent, (X_1, X_2, X_4, X_5) would indeed be inducing.

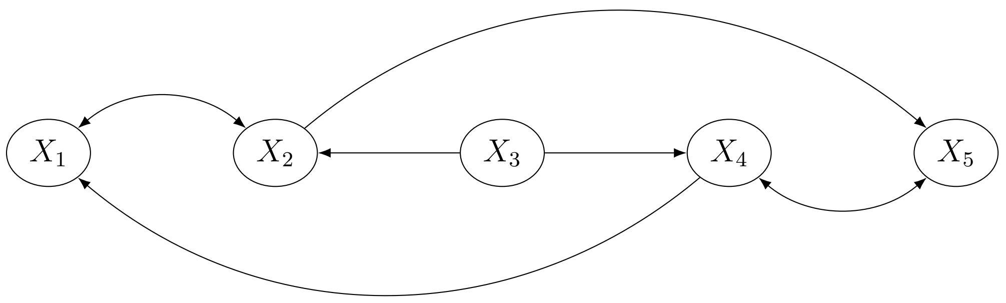

Figure 2.15.: Example of two variables not being blocked by adjacent nodes ([Spi+00] Section 6.3)

But here X_3 is observed, and while not being in adj_{X_1} or adj_{X_5}, serves the key role of being able to m-separate X_1 and X_5 to make the graph a MAG: $X_1 \perp\!\!\!\perp X_2 \mid X_2, X_3, X_4$. With X_3 together, the collider path $X_1 \leftrightarrow X_2 \leftarrow X_3 \rightarrow X_4 \leftrightarrow X_5$ that was opened by conditioning on X_2, X_4 is blocked again by X_3.

This has the consequence for the FCI algorithm that one needs to look beyond adjacencies for testing potential separating sets.

Definition 2.39 (**Possible d-separating set** [Col+12a]). *$X \in pds(X_i, X_j)$ if and only if $X \notin \{X_i, X_j\}$ and there is a path π between X_i and X in G such that for every subpath (X_k, X_l, X_m) of π either X_l is a collider on π or X_k and X_m are adjacent.*

The idea behind $pds(X_i, X_j)$ is as with the minimum neighbor separator and parents blocking in PC: if an m-separating set exists, it must be in $pds(X_i, X_j)$, so it suffices to search for separating sets there. There are other definitions of possible d-separating sets, [Col+12a] discusses some of them. Further explanation on possible d-separating sets can also be found in [Spi+00] Section 6.7. As this is not essential to this thesis, I will not elaborate further.

The FCI algorithm starts with a fully connected graph, where all edges are $\circ\!\!-\!\!\circ$ to show prior uncertainty about the nature of the edge marks. The first part is like the skeleton discovery phase of the PC algorithm, followed by unshielded collider orientation. Since not all separable nodes can be separated this way, in the second phase the possible d-separating sets are considered and tested whether more conditional independencies can be found. Lastly, it requires the orientation rules R0-R10 (stated in [Zha08b]) for completeness, as was shown by [Zha08b].

Algorithm 3 FCI algorithm

```
Require: Joint distribution $P_O$ of $d$ observed variables, independence oracle
 1: form complete graph $G = (O, E)$ on vertex set $O$ with edges $\circ\!\!-\!\!\circ$
 2: for $k = 0, \dots, d-2$ do                                    ▷ find skeleton
 3:     for $X_i \in V$ with $|adj^G_{X_i}| > k$ do
 4:         for $X_j \in adj^G_{X_i}$ do
 5:             for all $S \subset adj^G_{X_i} \setminus X_j$ with $|S| = k$ do
 6:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
 7:                     delete edge $X_i \circ\!\!-\!\!\circ X_j$ from $G$
 8:                     $S_{i,j} = S_{j,i} = S$
 9: for all unshielded triples $(X_i, X_j, X_k)$ do
10:     if $X_j \notin S_{i,k}$ then
11:         orient $X_i *\!\!-\!\!\circ X_j \circ\!\!-\!\!* X_k$ as $X_i *\!\!\rightarrow X_j \leftarrow\!\!* X_k$ in $G$
12: for all nodes $X_i$ in $G$ do
13:     compute $pds(G, X_i, \cdot)$ as defined in Definition 2.39
14:     for all nodes $X_j \in adj^G_{X_i}$ do
15:         for $k = 0, ..., d-2$ do
16:             for $|S| \subset pds(G, X_i, \cdot) \setminus \{X_j\}$ with $|S| = k$ do
17:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
18:                     delete edge $X_i *\!\!-\!\!* X_j$ from $G$
19:                     let $S_{i,j} = S_{j,i} = S$
20: reorient all edges in $G$ as $\circ\!\!-\!\!\circ$
21: use rules R0-R10 of [Zha08b] to orient as many edge marks as possible
22: return the maximally informative PAG $G = (V, E)$
```

2.4. Conclusion

This chapter introduced the relevant statistical theory for graphs, causality and causal discovery. It introduced DAGs and d-separation, as well as concepts to relate graphs to probability distributions. Latent variables were discussed and an overview on causal discovery, with an in-depth explanation of the constraint based methods PC and FCI was given. The next Chapter 3 introduces background knowledge and C-DAGs in particular, to then later be able to combine C-DAGs with PC and FCI in Chapters 4 and 5.

3. Overview on background knowledge frameworks for causal discovery

As seen in the previous chapter, causal discovery methods with observational data are limited by the Markov equivalence class (cf. Definition 2.13), which can be quite large, especially for sparse graphs. When the graph is not sparse, the search space becomes more difficult to traverse. In addition, causal discovery methods tend to scale poorly with increasing number of variables, as the space of DAGs grows superexponentially with the number of nodes. This is a problem for score based search algorithms like GES [Chi02], because they need to traverse that search space efficiently. Continuous score based methods like DAGs with NO TEARS [Zhe+18] sometimes perform better, but that may be due to problems with simulated data, as shown in [RSW21], and not their superior design.

Constraint-based methods like PC rely on conditional independence tests (CI tests). The number of them that need to be performed grows rapidly with the number of nodes and edges that a graph has. Furthermore, CI tests fail more the less data is available and the bigger the conditioning set is. This can lead to a cascading problem, since efficient implementations of PC search for separating sets between X, Y in nb_X, nb_Y. If a neighbor is falsely removed due to an erroneous independence test, the algorithm might no longer be able to find a separating set and then fail to remove other edges. In that way, errors can propagate. This risk becomes higher the more variables the graph has.

It is therefore desirable to improve these methods. Fortunately, one often has some idea on what the target graph is like, but these are not enough to specify the entire DAG confidently (otherwise one would directly move to causal inference on the constructed DAG). The idea is to formalize the prior expert knowledge into constraints on the graph and combine them with a causal discovery algorithm. The constraints could be a temporal ordering of the variables, knowledge about ancestral relationships (or lack thereof) and requiring/forbidding certain edges. Prior knowledge is also called background knowledge, and ongoing research effort is put into developing ways to incorporate it into causal discovery algorithms.

Background knowledge has been discussed to actively guide score based methods like KGS [HG23], A^*-based methods [KLC22] and NOTEARS [CRT23]. For constraint-based methods, except for [ASC20], background knowledge is usually used to orient additional edges after receiving the CPDAG ([Bro+22], [BD23]) and not to guide the edge removal phase to be more efficient.

So background knowledge is applicable when one has access to partial knowledge about the data generating process, which means parts of the DAG can be pre-specified (due to domain experts or previously performed experiments). There are two main ideas of how background knowledge can be described, which I will call aggregate background knowledge and pairwise background knowledge. Pairwise background knowledge ([Fan+22], [Mee13]) takes the form of restricting the relationship between pairs of variables.

Definition 3.1 (Pairwise background knowledge [Fan+22]). *For a graph $G = (V, E)$ pairwise background knowledge is a tuple $B_{pair} := (R, F)$ with R containing edges of the form $X_i \to X_j$, requiring a parental connection or $X_i \dashrightarrow X_j$, requiring an ancestral connection. Similarly, F contains edges of the form $X_i \not\to X_j$, forbidding a parental connection or $X_i \not\dashrightarrow X_j$, forbidding an ancestral connection.*

The definition of pairwise background knowledge from other authors sometimes does not include ancestral constraints (e.g. [Mee13]). Note also that X_i being a father of X_j ($X_i \to X_j$) implies X_i being an ancestor of X_j ($X_i \dashrightarrow X_j$) and $X_i \not\dashrightarrow X_j$ imples $X_i \not\to X_j$.

Aggregate background knowledge does not follow a uniform definition and several variants have been proposed; nevertheless, it is sometimes possible to represent them in terms of pairwise background knowledge and compare them via this route. The frameworks I will focus on here are Cluster-DAGs (C-DAGs), tiered background knowledge (tbk, [ASC20] [BD23]) and typing assumption [Bro+22]. Their common idea is to define relationships between groups of variables, which leads to an equivalence class of possible DAGs. This means that results from causal interpretation of CPDAGs and chain graphs ([LR02]) with the IDA algorithm (Intervention when DAG is Absent, [MKB09]) can sometimes also find applications.

C-DAGs are a special form of aggregate background knowledge as they allow for causal inference on the background knowledge alone already. In the original paper of [Ana+23] C-DAGs were used for this and not to enhance causal discovery. But C-DAGs are also very interesting for causal discovery as they unite several benefits:

- C-DAGs are easy to read as they are a macro-graph,
- C-DAGs allow for more customization between groups compared to other methods (cf. Sections 3.2 and 3.3),
- C-DAGs provide a structure for warm-starting constraint based discovery algorithms like PC and FCI.

This chapter will first discuss C-DAG terminology and properties in Section 3.1, as well as expand on the previous bullet points about why I think C-DAGs are a particularly interesting framework for describing background knowledge. Then I will discuss tiered background knowledge in Section 3.2 and in Section 3.3 I discuss the typing assumption from [Bro+22]. Both of these settings are aggregate background knowledge frameworks related to C-DAGs and therefore I will show their similarities and differences. Additionally, I discuss how each of them can be represented as boolean combinations of pairwise background knowledge. This embeds all of these methods into the framework of [ASC20] with its theoretical results on the unification of pairwise representation of causal background knowledge. Additionally, it makes comparisons of methods easier. How to adapt C-DAGs to causal discovery is discussed in Chapters 4 and 5. Section 3.4 concludes.

To round up this introduction on background knowledge, for readers interested in how to decide which background knowledge should be added to causal discovery, the causal knowledge hierarchy (CKH) framework [Adi+22] could be useful. It contains three tiers of knowledge, analogous to the "hierarchy of evidence" from [Ack+08]. The first, second and third tier are causal knowledge from expert opinion, from data and from peer-reviewed literature, respectively. They suggest that the target SCM should then be a convex combination of these three tiers. Another review of types of causal background knowledge and their impact on causal discovery is available in [CGK23], which partially overlaps with my discussion of pairwise and aggregate background knowledge, as well as some for this thesis less relevant forms of background knowledge, like requiring graph connectedness.

3.1. Cluster DAGs and cluster ADMGs

C-DAGs (Cluster-DAGs) were introduced in [Ana+23]. They are a form of background knowledge which aggregates variables into clusters and then defines macro-relationships between the clusters. The connection between variables in the same clusters gets no assumption, while a connection $C_1 \to C_2$ for vertices $X_1 \in C_1, X_2 \in C_2$ means there may or may not be an edge between X_1 and X_2, and if there is one, it would be of the form $X_1 \to X_2$.

The original C-DAG paper [Ana+23] defined the C-DAG w.r.t. the true underlying ADMG. Since I will analyze these structures both with and without bidirected edges, I decided to split the original definition and introduce the C-ADMG (Cluster acyclic directed mixed graph). The C-ADMG in Definition 3.2 corresponds to the original C-DAG definition, and when I am talking about a C-DAG, the cluster graph does not contain any bidirected edges, cf. Definition 3.3.

Definition 3.2 (**C-ADMG** [Ana+23])**.** *Given an ADMG $G = (V, E)$ (with directed and bidirected edges) and a partition $C = C_1, ..., C_k$ of V (i.e. $C_i \cap C_j = \emptyset$ for all $i \neq j$ and $V = \bigcup_{i=1}^k C_i$), construct a graph $G_C = (C, E_C)$ over C with a set of edges E_C defined as follows:*

(i) An edge $C_i \to C_j$ is in E_C if $\exists X_i \in C_i, X_j \in C_j$ such that $X_i \to X_j \in E$.

(ii) A bidirected edge $C_i \leftrightarrow C_j$ is in E_C if $\exists X_i \in C_i, X_j \in C_j$ such that $X_i \leftrightarrow X_j \in E$.

If the graph G_C contains no directed cycles, C is called an admissible partition of V and G_C is called a C-ADMG compatible with G. Any ADMG G that can imply G_C is called compatible with G_C.

Definition 3.3 (C-DAG). *A C-ADMG that has no bidirected edges in E_C is called a C-DAG.*

Arrows between vertices in the same cluster will be swallowed in that cluster and not have any effect on the C-ADMG. This means that an ADMG can still be projected to a C-DAG, as long as all bidirected edges fall into the same cluster and are therefore absorbed away. But usually when talking about C-DAGs I assume absence of all latent confounding, i.e. also absence of latent variables between nodes in the same cluster.

In the following, I might sometimes only mention C-DAGs or C-ADMGs for easier readability. Because C-DAGs are a special case of a C-ADMG, results that hold for C-ADMGs also hold for C-DAGs, and results that do not hold for C-DAGs then also do not hold for C-ADMGs.

3.1.1. Properties of C-ADMGs

By definition, C-ADMGs do not contain any cycles, thus standard graphical tools for causal inference are applicable [Ana+23]. For the purpose of causal discovery, while very helpful, this assumption isn't strictly necessary, as long as the true graph is acyclic (cf. [Bro+22] who allow cycles in the typing assumption).

Definition 3.2 is a graph over macro variables C and represents an equivalence class of ADMGs (all ADMGs that could imply the same C-ADMG). The analysis in this section follows [Ana+23] and shows how properties of ADMGs carry over to C-ADMGs. The first theorem shows what kind of m-separation represented by G_C carry over to compatible G's. The symbol $*$ represents a wildcard for either an arrowhead or a tail.

Definition 3.4 (M-separation in C-ADMGs [Ana+23]). *A path p in a C-ADMG G_C is said to be m-separated by a set of clusters $Z \subset C$ if and only if p contains a triplet*

(i) $C_i *\!\!-\!\!* C_m \to C_j$ such that the non-collider cluster C_m is in Z, or

(ii) $C_i *\!\!\to C_m \leftarrow\!\!* C_j$ such that the collider cluster C_m and its descendants are not in Z.

A set of clusters Z is said to m-separate two sets of clusters $X, Y \subset C$, denoted by $X \perp\!\!\!\perp_{G_C} Y \mid Z$, if and only if Z blocks every path from a cluster in X to a cluster in Y.

Theorem 3.5 (Soundness and completeness of m-separation in C-ADMGs [Ana+23]). *In a C-ADMG G_C, let $C_x, C_y, C_z \subset C$ be sets of clusters. If C_x and C_y are m-separated by C_z in G_C, then in any ADMG G compatible with G_C, C_x and C_y are m-separated by C_z in G, i.e.*

$$C_x \perp\!\!\!\perp_{G_C} C_y \mid C_z \Rightarrow C_x \perp\!\!\!\perp_G C_y \mid C_z \tag{3.1}$$

If C_x and C_y are not m-separated by C_z in G_C, then there exists an ADMG G compatible with G_C where C_x and C_y are not m-separated by C_z in G.

Proof. See [Ana+23]. □

C_x, C_y, C_z technically are unions of clusters, for example $C_x = \{\{X_1, X_2\}, \{X_3, X_4\}\}$, but for ease of notation, they can also be read as $C_x = \{X_1, X_2, X_3, X_4\}$ when they are referring to the original graph G and the original nodes V.

Corollary 3.6 (**M-separation of subsets**). *For any subsets $S_x \subset C_x$ and $S_y \subset C_y$ it holds that*

$$C_x \perp\!\!\!\perp_G C_y \mid C_z \Rightarrow S_x \perp\!\!\!\perp S_y \mid C_z \tag{3.2}$$

Proof. Apply the decomposition property from 2.10 twice for functions $h_x(C_x) = S_x, h_y(C_y) = S_y$, which are both measurable w.r.t. the power set as the σ-algebra. □

Corollary 3.6 implies that for m-separated clusters, any subset of the members of the separated clusters is also separated by the same set. For example, if $\{X_1, X_2\} \perp\!\!\!\perp_{G_C} \{X_3\} \mid \{X_4\}$ in G_C, then $X_1 \perp\!\!\!\perp_G X_3 \mid X_4$ in G for all G compatible with G_C. This is crucial to be able to delete edges absent from a C-DAG for causal discovery.

The distribution $P(v)$ associated with the ADMG G also factorizes over a compatible C-ADMG G_C:

Theorem 3.7 (**C-ADMG factorization property** [Ana+23]). *Let G_C be a C-ADMG compatible with ADMG G. If the observational distribution $P(v)$ factorizes according to G, then the observational distribution $P(v) = P(c)$ over macro variables C factorizes according to G_C:*

$$P(c) = \sum_u P(u) \prod_{k:C_k \in C} P(c_k \mid pa^G_{C_k}, u'_k) \tag{3.3}$$

where $U'_k \subset U$, the latent variables, such that for any i, j, $U'_i \cap U'_j \neq \emptyset$ if and only if there is a bidirected edge $C_i \Longleftrightarrow C_j$ in G_C.

Proof. See [Ana+23] □

It follows that if $P(v)$ factorizes according to DAG G and if the cluster graph G_C is a C-DAG (i.e. without bidirected edges) compatible with G, $P(c)$ satisfies the global Markov property w.r.t. G_C:

Corollary 3.8 (**Global Markov property for C-DAGs**). *Let G_C be a C-DAG compatible with a DAG G and let $P(v)$ satisfy the global Markov property w.r.t. G. Then the distribution over macro variables C, $P(c)$, satisfies the global Markov property w.r.t. G_C:*

$$P(v) \textit{ Markov w.r.t. } G \Rightarrow P(c) \textit{ Markov w.r.t } G_C \tag{3.4}$$

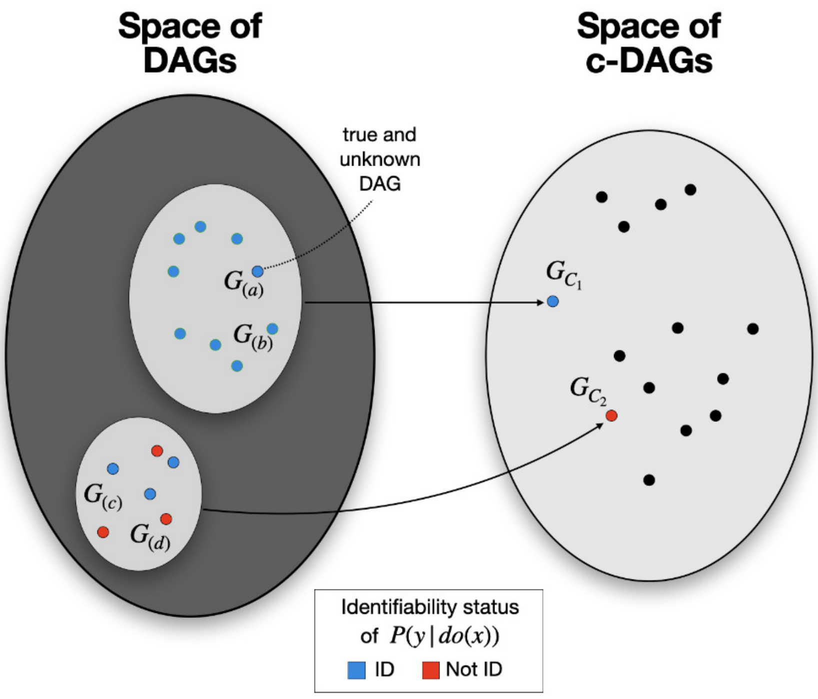

Figure 3.1.: Identifying $P(y|do(x))$ in a C-ADMG means identifying such an effect for the entire equivalence class. If the effect is not identifiable in at least one member of the equivalence class, it is not identifiable in the C-ADMG (Figure from [Ana+23]).

Proof. By Theorem 2.12 if $P(v)$ satisfies the global Markov property w.r.t. G, $P(v)$ also factorizes according to G. By Theorem 3.7 if $P(v)$ factorizes according to G, $P(c)$ factorizes according to G_C. And finally by Theorem 2.12 again, if $P(c)$ factorizes according to G_C, then it also satisfies the global Markov property w.r.t. G_C. In essence it is

$$\begin{aligned}&P(v) \text{ Markov w.r.t. } G \Rightarrow P(v) \text{ factorizes according to } G\\ &\qquad \Rightarrow P(c) \text{ factorizes according to } G_C \Rightarrow P(c) \text{ Markov w.r.t. } G_C. \qquad (3.5)\end{aligned}$$

□

These and further properties derived by [Ana+23] make it possible to use do-calculus [Pea09] and the ID-Algorithm ([SP06], [BP16]) in C-ADMGs. Theorem 3.5 is of particular importance, as it means that any ADMG G compatible with a C-ADMG G_C *inherits* all m-separations from G_C. Unfortunately, not all distributions are identifiable from the C-ADMG alone (cf. Figure 3.1), either due to inducing paths in the C-ADMG or because the two variables of interest are in the same cluster.

To see how inducing paths make adjustment difficult, consider the C-ADMG in Figure 3.2 from [Ana+23] with variables $\{X, Y, Z_1, Z_2, Z_3\}$ and clusters $C = \{C_1, C_2, C_3\}, C_1 = \{X\}, C_2 = \{Z_1, Z_2, Z_3\}, C_3 = \{Y\}$. It has the inducing path $C_1 \leftrightarrow C_2 \leftrightarrow C_3$ because C_2 is an ancestor of C_3.

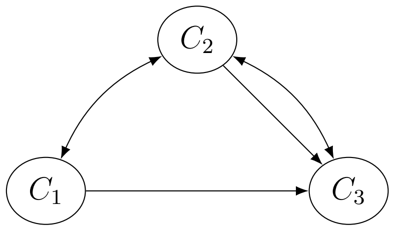

Figure 3.2.: Example of C-ADMG G_C.

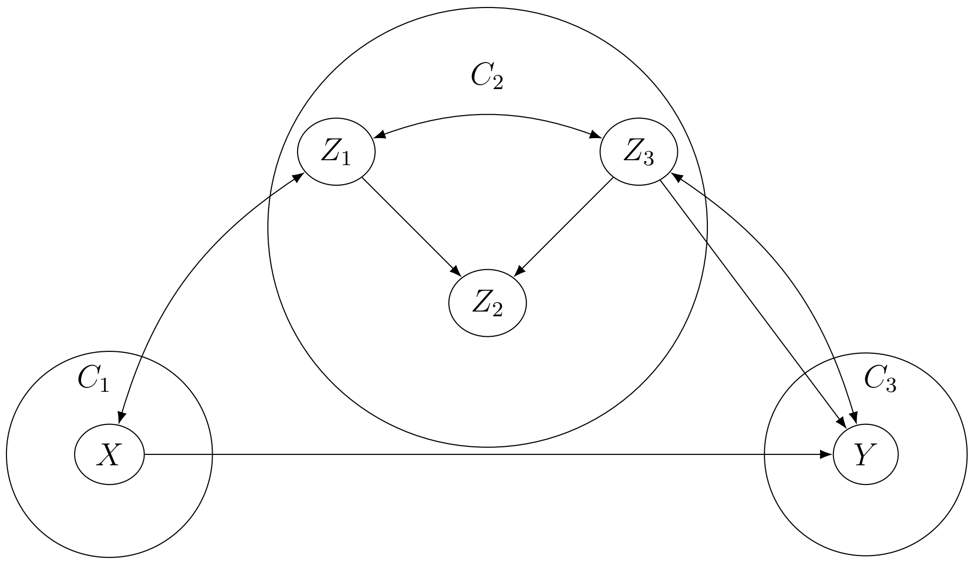

Figure 3.3.: G_1 compatible with G_C of Figure 3.2.

Identifying the effect $P(y|do(x)) = P(C_3|do(C_1))$ is not possible from G_C (see [Pea09] for the rules of do-calculus). G_C can not decide whether $P(C_3|do(C_1)) = P(C_3|C_1)$, $P(C_3|do(C_1)) = P(C_3|C_1, C_2)$, it may even be that both of these are not correct, i.e. cluster adjustment is not possible. For the graph G_1 in Figure 3.3, which is compatible with G_C, $P(C_3|do(C_1)) = P(C_3|C_1)$ would be correct. This is because all paths going through C_2 in G_1 are collider paths and are blocked by not conditioning on C_2. However, the graph G_2 in Figure 3.4 is also compatible with G_C. In that graph, $P(C_3|do(C_1)) = P(C_3|C_1)$ fails due to the path $X \leftrightarrow Z_1 \rightarrow Z_2 \rightarrow Y$ not being blocked and $P(C_3|do(C_1)) = P(C_3|C_1, C_2)$ fails due to the path $X \leftrightarrow Z_1 \leftrightarrow Z_3 \leftrightarrow Y$ being opened up by conditioning on colliders Z_1, Z_3. For G_2, cluster adjustment is undecidable and hence the distribution $P(Y|do(X))$ is unidentifiable.

Thus if there is at least one ADMG G compatible with C-ADMG G_C where a distribution is not identifiable, the distribution is not identifiable from the C-ADMG alone. In this case, one then would like to use causal discovery to, for example, figure out the structure among the variables in cluster C_2 (Figure 3.2). Afterwards, one then can successfully identify causal effects. The C-ADMG already contains some useful information, so it shouldn't be disregarded altogether. This motivates the causal discovery with C-DAGs

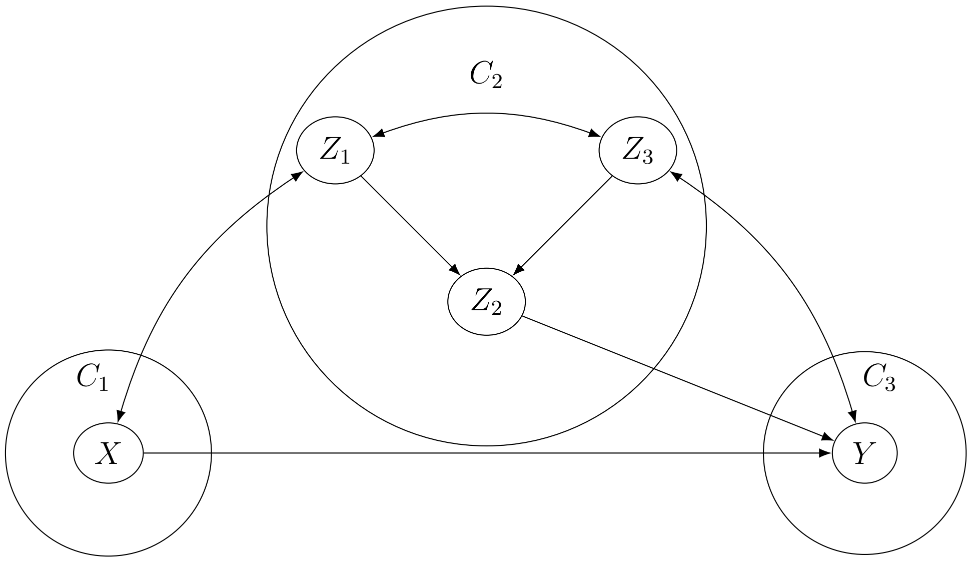

Figure 3.4.: G_2 compatible with G_C of Figure 3.2.

and C-ADMGs and will be discussed in Chapters 4 and 5. Before moving on to these parts, I first want to consider how one can straightforwadly construct a C-ADMG.

3.1.2. Construction of C-ADMGs

Construction of C-ADMGs can become difficult when the true DAG is unkown (as would be the case in most practical applications) and the clustering partition is unclear. After finishing cluster graph construction, one may realize it unintentionally contains a cycle or is not as expressive as was wished for. In this section I present results that can guide the clustering process to avoid such issues. To start, note that not every partition of V will be admissible. Consider the DAG and clustering in Figure 3.5.

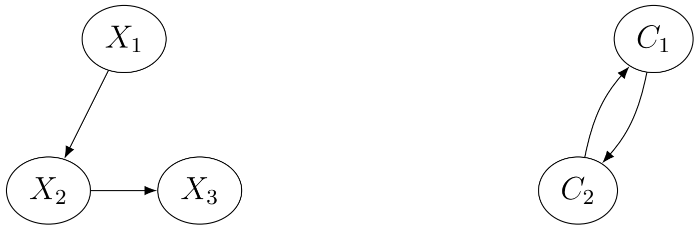

Figure 3.5.: Example for inadmissible clustering $C = \{C_1, C_2\}$, $C_1 = \{X_1, X_3\}, C_2 = \{X_2\}$.

It is clear that C is a partition of V, but the cluster graph G_C is not acyclic. The problem comes from the fact that C_1 clusters together X_1 and X_3, while X_1's descendants and X_3's ancestors overlap, and that overlap X_2 is not included in C_1. This will lead to an edge coming from X_2's cluster C_2 into C_1, as well as an edge going into X_2's cluster C_2 from C_1, a cycle.

A naive solution to this problem is to just decide on some partition, construct G_C, check for acyclicity, and repeat until the cluster graph is admissible. There is however a more principled method to ensure acyclicity, even when the DAG is unknown as long as the prior knowledge is used correctly.

First some additional notation is needed to concisely describe vertice relationships across the two graphs G and G_C, which expands on the notation in Definition 2.2. The following analysis has to reason about sets such as $\bigcup_{v \in C} pa_v \setminus C$, representing the set of parents of vertices $X \in C$ *outside* cluster C in the Graph G.

Definition 3.9 (**Additional node relations for cluster graphs**). *To reason about node relationships of clusters in the original graph, I will use the following definitions for some cluster $C_i \in C$, where C is a partition.*

- $pa^G_{C_i} := \bigcup_{C_k \in pa^{G_C}_{C_i}} C_k$ *describes the parents of C_i in G, outside of C_i.*
- $ch^G_{C_i} := \bigcup_{C_k \in ch^{G_C}_{C_i}} C_k$ *describes the children of C_i in G, outside of C_i.*
- $an^G_{C_i} := \bigcup_{C_k \in an^{G_C}_{C_i}} C_k$ *describes the ancestors of C_i in G, outside of C_i.*
- $de^G_{C_i} := \bigcup_{C_k \in de^{G_C}_{C_i}} C_k$ *describes the descendants of C_i in G, outside of C_i.*

Note that $pa^G_{C_i}$ and $pa^{G_C}_{C_i}$ are not the same. The difference in subscript G and G_C shows which vertice set it relates to. $pa^G_{C_i}$ as defined above describes vertices in G, the parents of the cluster members, while $pa^{G_C}_{C_i}$ describes clusters in G_C, the parents of C in the cluster graph. Note that $C_i \not\subset pa^G_{C_i}$, as C_i can not be its own parent in G_C. The same holds true for *ch*, *an* and *de*.

This definition may seem unintuitive at first glance: it is possible for X_j to be in $pa^G_{C_i}$ even though for no $X_i \in C_i$ an edge $X_j \rightarrow X_i$ exists. It suffices that for another vertice Y_j in X_j's cluster C_j an edge $Y_j \rightarrow X_i$ exists. In the same way, it is possible that $Y \rightarrow X$, $X \in C_i$ but $Y \notin pa^G_{C_i}$. Since $C_i \not\subset pa^G_{C_i}$, if $Y \in C_i$ it will not be included in $pa^G_{C_i}$ according to Definition 3.9. As an example, see Figure 3.6. $X_1 \in pa^G_{C_2}$ but $X_1 \notin pa^G_X$ for any $X \in C_2$. Furthermore $X_3 \in pa^G_{X_4}, X_4 \in C_2$ but $X_3 \notin pa^G_{C_2}$.

It may seem strange that X_1 is in the parents of C_2 ($X_1 \in pa^G_{C_2} = \{X_1, X_2\}$), even though X_1 is not a parent of any member of C_2. But in order to avoid cycles in G_C, it is important for X_1 to be in $pa^G_{C_2}$. Consider that the cluster $C_1 = \{X_1, X_2\}$ by Definition 3.2 merges together the ancestors and descendants of its members X_1, X_2. This means directed paths into X_1 (X_2) get merged together with directed paths going out of X_2 (X_1). For example, the addition of edge $X_4 \rightarrow X_1$ would not introduce a cycle in G, but in G_C there would now be a cycle. The directed path $X_4 \rightarrow X_1$, which is into X_1, gets merged together with the directed path $X_2 \rightarrow X_3$, which is out of X_2. On C-DAG level, that means there is a directed path $C_2 \rightarrow C_1$ due to $X_4 \rightarrow X_1$ and a directed

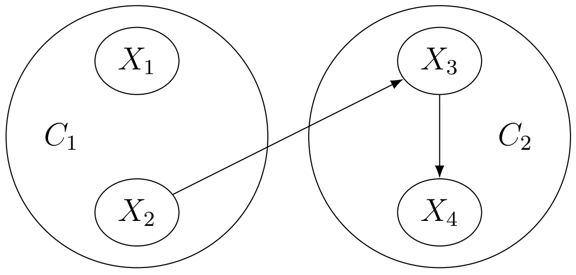

Figure 3.6.: DAG G and C-DAG G_C with $C = \{C_1, C_2\}, C_1 = \{X_1, X_2\}, C_2 = \{X_3, X_4\}$

path $C_1 \to C_2$ due to $X_2 \to X_3$. This is a cycle in G_C. So to reason about cycles, the definitions need to be constructed as in Definition 3.9 and thus argumentation in the G domain over potential clusters can guide the clustering process.

This allows equivalent conditions for admissible clusterings in the following theorem.

Theorem 3.10 (Necessary and sufficient conditions for clustering admissibility). *Given an ADMG G and a partition $C = \{C_1, ..., C_m\}$, the following three statements are equivalent:*

(i) The C-ADMG G_C is admissible (G_C contains no cycles).

(ii) For all $i : C_i \not\to C_i$ and for all $i \neq j$: $de^G_{C_i} \cap C_j = \emptyset$ or $an^G_{C_i} \cap C_j = \emptyset$.

(iii) For all $i : C_i \not\to C_i$ and $de^G_{C_i} \cap an^G_{C_i} = \emptyset$.

Proof. Let C contain at least two clusters C_i, C_j with $C_i, C_j \neq \emptyset$, else the proof is trivial.

"$(i) \Rightarrow (ii)$": Assume C is admissible, i.e. G_C has no cycles. First, for all C_i, admissibility implies no cycles, hence $C_i \not\to C_i$. Let $C_i, C_j \in C$ be two clusters. By acyclicity, either $C_j \subset an^G_{C_i}$, $C_j \subset de^G_{C_i}$ or $C_j \not\subset an^G_{C_i}$ and $C_j \not\subset de^G_{C_i}$. In the first case, due to acyclicity $de^G_{C_i} \cap C_j = \emptyset$, in the second case $an^G_{C_i} \cap C_j = \emptyset$ and in the third case $de^G_{C_i} \cap C_j = \emptyset$ and $an^G_{C_i} \cap C_j = \emptyset$.

"$(ii) \Rightarrow (iii)$": $C_i \not\to C_i$ carries over. For any $C_i \in C$, it is $an^G_{C_i} = \bigcup_{C_k \in an^{G_C}_{C_i}} C_k$ and for all of these $C_k \in an^{G_C}_{C_i}$ it is $an^G_{C_i} \cap C_k = \bigcup_{C_l \in an^{G_C}_{C_i}} C_l \cap C_k = C_k \neq \emptyset$. Thus $de^G_{C_i} \cap C_k = \emptyset$ for those $C_k \in an^{G_C}_{C_i}$ by assumption (ii). To conclude, see that

$$an^G_{C_i} \cap de^G_{C_i} = \bigcup_{C_k \in an^{G_C}_{C_i}} (C_k \cap de^G_{C_i}) = \emptyset.$$

"$(iii) \Rightarrow (i)$": Proof by contraposition. Assume G_C contains a cycle. If the cycle is of the form $C_i \to C_i$, the proof is finished. Otherwise let the cycle be of the form $C_i \to C_j \to ... \to C_i$ and one has to show that $de^G_{C_i} \cap an^G_{C_i} \neq \emptyset$. Looking at the path $C_i \to C_j \to ... \to C_i$, it must be $C_j \subset an^G_{C_i}$ and $C_j \subset de^G_{C_i}$ and therefore $\emptyset \neq C_j \subset de^G_{C_i} \cap an^G_{C_i} \neq \emptyset$. This finishes the contraposition.

□

Theorem 3.10 (iii) gives a graphical criterion that can be checked algorithmically whether G_C contains a cycle and Theorem 3.10 (ii) gives a hint on what to look for when constructing a clustering iteratively. When one thinks of adding a cluster C_j, if for another already existing cluster C_i it is $de^G_{C_i} \cap C_j \neq \emptyset$ and $an^G_{C_i} \cap C_j \neq \emptyset$, then C_j would introduce a cycle. This means one cannot proceed and has to change something about either C_i or C_j.

A property of admissible clusterings is that for vertices in the same cluster, directed paths between them get contracted into that cluster:

Theorem 3.11 **(Causal contraction).** *An admissible clustering C has the property that for any C_i and directed path π between nodes $X_1, X_2 \in C_i$, for all $Y \in \pi$ it follows that $Y \in C_i$.*

Proof. Proof by contraposition, assume there was a path $X_1 \to ... \to Y \to ... \to X_2$ with $X_1, X_2 \in C_i$ but $Y \notin C_i$, say $Y \in C_j$. This means in G_C there would be a path of the form $C_i \to ... \to C_j \to ... \to C_i$, a cycle, in contradiction to the admissibility of C. □

The theorem above gives another hint on how to construct a cluster: if one wants to cluster vertices together that are connected by a directed path (or when the ADMG is unknown, one suspects that there could be a directed path between those variables), all member of that (suspected) directed path should also be clustered into the same cluster.

This however is not enough, as when clustering, 'phantom' directed paths could be created, as seen in Figure 3.6. These phantom paths also need to be taken into account. In practice C-ADMGs will be constructed iteratively. First one cluster is specified, then another, then some variables are shuffled around until all variables are assigned to a cluster. This can lead to inadmissible clusterings, as a cluster that was specified earlier could suddenly become part of a cycle when shuffling around variables in other clusters or when a new cluster is introduces (Figure 3.6).

Consider the graph in Figure 3.7 with the clustering procedure as follows: first $C_1 = \{X_1, X_5\}$ is specified, then $C_2 = \{X_2, X_3\}$. Up until now there are no cycles, but X_4 is unassigned and let's say it gets added to C_2. Now there is a cycle $C_1 \to C_2 \to C_1$, even though clustering only with C_1 or C_2 (or C_1 and $C_2 = \{X_2, X_3\}$ without X_4) was completely fine.

So the partition $C = \{C_1, C_2\}$ would be inadmissible, even though the previous partitions $C_a = \{C_1, \{X_2\}, \{X_3\}, \{X_4\}\}$ and $C_b = \{\{X_1\}, C_2, \{X_5\}\}$ were admissible clusterings, as can be seen in Figure 3.8. What went wrong?

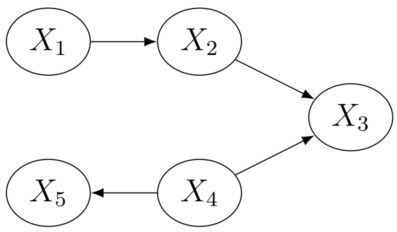

Figure 3.7.: Example graph for an erroneous clustering procedure.

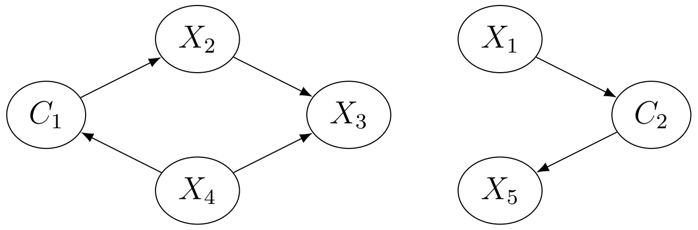

Figure 3.8.: C_1 and C_2 alone would not introduce cycles.

In G of Figure 3.7, there exists no directed path from X_1 to X_5, i.e. $X_1 \notin an^G_{\{X_5\}}$. However, $X_1 \in an^{G_{C_b}}_{\{X_5\}}$, because that clustering introduces the directed path $\{X_1\} \to C_2 \to \{X_5\}$ in G_{C_b}. Therefore, should one want to cluster X_1, X_5 together in G_{C_b}, the causal contraction property of admissible clusterings from Theorem 3.11 would be violated. This problem comes from the fact that Theorem 3.10 (ii) and (iii) are violated: $de^G_{C_1} \cap C_2 = C_2 \neq \emptyset$ and $an^G_{C_1} \cap C_2 = C_2 \neq \emptyset$ violates Theorem 3.10 (ii) and $an^G_{C_1} \cap de^G_{C_1} = C_2 \neq \emptyset$ violates Theorem 3.10 (iii).

It would be beneficial if it was possible to construct a cluster (in the above example C_1) that would produce an admissible clustering together with *any* admissible partition of the remaining variables. More formally this means: If $C_1 = \{X_1, ..., X_{i-1}\}$ and $C_a = \{C_1, \{X_i\}, ...\{X_n\}\}$ is an admissible clustering, one would like $C_b = \{C_1, C_2, ..., C_m\}$ to be admissible for any partition $\{C_2, ..., C_m\}$ of $V \setminus C_1$ if the partition $\{\{X_1\}, ..., \{X_{i-1}\}, C_2, ...C_m\}$ is admissible.

If that were the case, one could specify C_1 and then specify the remaining clusters iteratively without having to ever check with the previous clusters whether cycles were introduced.

Even more importantly, if the true ADMG is unknown, there should be a clear procedure on how the prior knowledge guides the clustering process, which then hopefully is less prone to the aforementioned problems.

Remember what went wrong in the example above: the clusters C_1, C_2 had overlapping ancestor and descendant sets. If one clusters along a (suspected) topological ordering and cluster variables together that either sit at the very top or the very bottom, for every cluster at the top (bottom) the ancestor (descendant) set is guaranteed be empty and therefore can guarantee an acyclic C-ADMG.

Theorem 3.12 (**Guaranteed admissible clustering**). *For a given ADMG $G = (V, E)$ and topologically ordered vertices $V = \{X_1, ..., X_d\}$ let a partition be defined as follows: $C_1 := \{X_1, ..., X_{j_1}\}, C_2 := \{X_{j_1+1}, ..., X_{j_2}\}, ..., C_m := \{X_{j_{m-1}+1}, ..., X_{j_m}\}$ with $j_1 \leq j_2 \leq ... \leq j_m$. Then the partition $C = \{C_1, ..., C_m\}$ is admissible and G_C is a C-ADMG.*

Proof. Proof by contraposition, if G_C was not acyclic, there would be a cycle $C_i \to C_i$ or $C_i \to ... \to C_k \to ... \to C_i$. Both cases are impossible, as it would imply for C_k there is a cluster C_i which contains vertices above and below C_k in the topological ordering, i.e. $\exists X_l, X_m \in C_i$ where $l < r$ and $r < m$ for all $X_r \in C_k$. This is in contradiction with the construction of C. W.l.o.g. if it was $i < k$, then $j_i \leq r \leq j_k \leq l, m$ but previously it also was $l < r$, a contradiction. $\square$

Motivated by Theorem 3.12 one starts to cluster either at the top or bottom of a topological ordering (which for a DAG or ADMG is guaranteed to exist, cf. Proposition 2.8) and repeats for the remaining variables. Edges between the clusters are set according to the underlying ADMG, which guaratnees acyclicity of the C-ADMG. This procedure is formalized in Algorithm 4. In that algorithm, a variable is considered to be clustered if it was added to one cluster in one of the 'if' clauses.

This procedure would avoid the issue encountered in Figure 3.7. Neither C_1 nor C_2 can be constructed in the first step, as they have non-empty ancestors and descendants that are not part of a cluster yet. Therefore say $\{X_1\}$ is made the first cluster. After that, C_2 can be established as a cluster from 'above', as $an_{C_2}^{G_{\tilde{C}}} = \{X_1\}$, which is a union of previous 'upper' clusters. Constructing cluster C_1 now is not possible: one would need to add X_5 to previously established cluster $\{X_1\}$, but $an_{\{X_5\}}^{G_C} \cap de_{\{X_1\}}^{G_C} = \{X_2, X_3, X_4\} \neq \emptyset$. This means $\{X_5\}$ has to become its own cluster from 'below'. The resulting C-ADMG now contains no cycles.

If one has access to the true ADMG G, the prior knowledge needed in Algorithm 4 is readable from that graph. In practice one wants to construct a C-ADMG for an unkown ADMG. In that case, the graph with its prior knowledge would be replaced by the assumptions about the variable ordering, i.e. create cluster C_i if its variables are assumed to fit well into that (topological) place in the resulting C-ADMG G_C and not introduce cycles. Edges would be placed where it is possible for at least one edge to be present between vertices of the two clusters.

3.1.3. Pairwise representation of C-DAGs and C-ADMGs

In this section I present how C-DAGs and C-ADMGs can be represented as a boolean combination of pairwise constraints. This is useful for comparison with other background knowledge frameworks and shows that all results from [Fan+22] also apply to C-DAGs and C-ADMGs, namely decomposition into an MPDAG with direct causal

Algorithm 4 C-ADMG construction

```
Require: Nodes in topological ordering $V = \{X_1, ..., X_d\}$, graph $G = (V, E)$
 1: Let $C := \{\{X_1\}, ..., \{X_d\}\}$ be the initial partition
 2: set $i, j = 0$
 3: while not all variables are clustered do one of the following:
 4:     if want new cluster 'above' then
 5:         $i \leftarrow i + 1$
 6:         update partition $C$ with $C_i$ to partition
            $\tilde{C} := \{C_1, ..., C_{i-1}, C_i, \{X_l\}, ..., \{X_k\}, C_{-j}, ..., C_{-1}\}$
 7:         if $an^{G_{\tilde{C}}}_{C_i} = \emptyset$ or $an^{G_{\tilde{C}}}_{C_i} = \bigcup_{m \in M} C_m$ of clusters with index set $M \subset [i-1]$ then
 8:             $C = \tilde{C}$
 9:             add directed edges to $G_C$ according to $G$
10:             add bidirected edges to $G_C$ according to $G$
11:         else
12:             go to line 3
13:     if want new cluster 'below' then
14:         $j \leftarrow j + 1$
15:         update partition $C$ with $C_{-j}$ to partition
            $\tilde{C} := \{C_1, ..., C_i, \{X_l\}, ..., \{X_k\}, C_{-j}, C_{-j+1}, ..., C_{-1}\}$
16:         if $de^{G_{\tilde{C}}}_{C_{-j}} = \emptyset$ or $de^{G_{\tilde{C}}}_{C_{-j}} = \bigcup_{m \in M} C_m$ of clusters with index set $M \subset [-j+1]$
            then
17:             $C = \tilde{C}$
18:             add directed edges to $G_C$ according to $G$
19:             add bidirected edges to $G_C$ according to $G$
20:         else
21:             go to line 3
22:     if want to add variable $X_l$ to previously established cluster $C_k$ then
23:         if $an^G_{\{X_l\}} \cap de^G_{C_k} = \emptyset$ and $de^{G_C}_{\{X_l\}} \cap an^G_{C_k} = \emptyset$ then
24:             add variable $X_l$ to cluster $C_k$ in new clustering $\tilde{C}$
25:             $C = \tilde{C}$
26:             add directed edges to $G_C$ according to $G$
27:             add bidirected edges to $G_C$ according to $G$
28:         else
29:             go to line 3
30: Merge clusters as desired, as long as directed paths between them get contracted
    (Theorem 3.11).
31: return $G_C$
```

clauses (DCCs), sufficient and necessary conditions to identify causal effects and a local IDA-algorithm to estimate values of an unidentifiable effect. Furthermore this allows one to understand the exact constraints a C-ADMG imposes on any pair of variables. To start, I focus on the case of C-DAGs and will discuss the bidirected C-ADMG case afterwards.

Recall Definition 3.1 from [Fan+22], where between two vertices X_i, X_j either no constraint, a parental constraint ($X_i \to X_j$ or $X_i \not\to X_j$) or an ancestral constraint ($X_i \dashrightarrow X_j$ or $X_i \not\dashrightarrow X_j$) can be imposed. If two nodes X_i, X_j are in the same cluster C_i, no pairwise assumption is made. Next, what does $X_i \in C_i, X_j \in C_j$ and $C_i \to C_j$ mean in pairwise language? The pairwise background knowledge constraint has to be true *for all* G that are compatible with G_C and a one-to-one correspondence between C-DAG and pairwise constraints would be desirable:

$$G \text{ compatible with } G_C \iff \text{pairwise constraints true for } G \tag{3.6}$$

That means that $X_i \to X_j$ cannot be the correct pairwise constraint, as nodes of adjacent clusters do not necessarily have to be adjacent. What $C_i \to C_j$ actually means is the absence of directed paths $X_i \leftarrow ... \leftarrow X_j$. Why? If there was such a path in G, it would correspond to a directed path $C_i \leftarrow ... \leftarrow C_j$ in G_C, which would indicate a cycle, in contradiction to the C-DAG acyclicity assumption.

Analogously, the other situations can also be described by pairwise constraints, which is formalized in the next theorem.

Theorem 3.13 (Pairwise characterization of C-DAGs). *Clusters C_i, C_j imply, depending on their relationship in C-DAG G_C, pairwise constraints in the following ways:*

(i)

$$C_i = C_j \Rightarrow \boldsymbol{T}$$

(the tautology, which is always true and places no restriction),

(ii)

$$C_i \to C_j \Rightarrow \bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\dashleftarrow X_j \wedge \bigvee_{X_i \in C_i, X_j \in C_j} X_i \to X_j =: dir(C_i, C_j)$$

(note that $X_i \not\dashleftarrow X_j$ also implies $X_i \not\leftarrow X_j$),

(iii)

$$C_i \dashrightarrow C_j \wedge C_i \not\to C_j \Rightarrow \bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\dashleftarrow X_j \wedge X_i \not\to X_j =: anc(C_i, C_j)$$

,

(iv)

$$C_i \not\dashrightarrow C_j \wedge C_i \not\dashleftarrow C_j \wedge C_i \neq C_j \Rightarrow \bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\dashleftarrow X_j \wedge X_i \not\dashrightarrow X_j =: nrel(C_i, C_j)$$

.

These four points exhaust all possibilities in which two clusters could relate to each other in a C-DAG. The background knowledge implied by C-DAG G_C with clustering $C = \{C_1, ..., C_m\}$ can therefore be represented in pairwise form:

$$bk(G_C) := \bigwedge_{C_i, C_j \in C} \Big(\bigwedge_{C_i \to C_j} dir(C_i, C_j) \wedge \bigwedge_{\substack{C_i \dashrightarrow C_j \\ C_i \not\to C_j}} anc(C_i, C_j) \wedge \bigwedge_{\substack{C_i \neq C_j, C_i \cancel{\dashrightarrow} C_j \\ C_i \cancel{\dashleftarrow} C_j}} nrel(C_i, C_j) \Big). \tag{3.7}$$

Furthermore for C-DAG G_C the following equivalence holds:

$$G \text{ compatible with } G_C \iff bk(G_C) \text{ is true for } G. \tag{3.8}$$

Proof. (i): For two nodes in the same cluster C_i, no restriction is made, so for all $X_i, X_j \in C_i$, $\mathbf{T}$ is true.
(ii): For two clusters C_i, C_j with $C_i \to C_j$ and nodes $X_i \in C_i, X_j \in C_j$, it is impossible to have $X_i \dashleftarrow X_j$, as that would mean $C_i \dashleftarrow C_j$ in G_C, which is a cycle together with $C_i \to C_j$ in contradiction to G_C being a C-DAG and C being admissible. In addition, at least one $X_i \to X_j$ needs to be present by C-DAG construction. Therefore $C_i \to C_j$ implies $\bigwedge_{X_i \in C_i, X_j \in C_j} X_i \cancel{\dashleftarrow} X_j \wedge \bigvee_{X_i \in C_i, X_j \in C_j} X_i \to X_j$.
(iii): Let two clusters C_i, C_j have $C_i \dashrightarrow C_j \wedge C_i \not\to C_j$, i.e. there is a directed path from one to another, but they are not adjacent. With nodes $X_i \in C_i, X_j \in C_j$ for analogous reasoning to (ii), it is impossible to have $X_i \dashleftarrow X_j$. In addition, as $C_i \not\to C_j$, it is also impossible to have $X_i \to X_j$. So $C_i \dashrightarrow C_j \wedge C_i \not\to C_j$ implies $X_i \cancel{\dashleftarrow} X_j \wedge X_i \not\leftarrow X_j$.
(iv): For two clusters that are not the same, not adjacent and not connected by a directed path, i.e. $C_i \cancel{\dashrightarrow} C_j \wedge C_i \cancel{\dashleftarrow} C_j \wedge C_i \neq C_j$ and nodes $X_i \in C_i, X_j \in C_j$ it is impossible for X_i, X_j to be connected by a directed path. Therefore this implies $\bigwedge_{X_i \in C_i, X_j \in C_j} X_i \cancel{\dashleftarrow} X_j \wedge X_i \cancel{\dashrightarrow} X_j$.

Equivalence statement:
"$\Rightarrow$": Let $G = (V, E)$ be a graph over the same variables V with edges E being compatible with C-DAG G_C. Furthermore let $bk(G_C)$ be as defined in Equation 3.8. The task is to show that $bk(G_C)$ is true for G, specifically, that any edge between any X_i, X_j satisfies $bk(G_C)$. Take any $X_i, X_j \in V$ and their respective clusters $X_i \in C_i, X_j \in C_j$. C_i, C_j relate to each other in exactly one of the four ways described in (i)-(iv). Whatever the cluster relationship is on the left hand side of (i)-(iv), the proofs of (i)-(iv) above show that the edge between X_i, X_j satisfies the constraint on the right hand side of (i)-(iv). So the boolean pairwise combination $bk(G_C)$ is true for G.
"$\Leftarrow$": Let G_C be a C-DAG and $bk(G_C)$ be true for DAG G. The task is to show that G is compatible with G_C. $G = (V, E)$ is compatible with G_C, if none of its edges E contradict the C-DAG edges E_C. Take any X_i, X_j and their corresponding clusters $X_i \in C_i, X_j \in C_j$. The edge between X_i, X_j can take any of the three forms $X_i \to X_j$, $X_i \leftarrow X_j$ or $X_i \cancel{-} X_j$. W.l.o.g. (no need to consider the case $C_i \leftarrow C_j$ due to symmetry) the C-DAG restriction on the edge between X_i, X_j can take any of the forms in (i)-(iv). If restriction $\mathbf{T}$ is put on the edge between X_i, X_j, they are put in the same cluster and

any of $X_i \to X_j$, $X_i \leftarrow X_j$ and $X_i \not\!\!- X_j$ would be compatible with G_C.
If restriction $X_i \not\dashleftarrow X_j$ is put, X_i and X_j are in adjacent clusters $C_i \to C_j$. The edges allowed by $X_i \not\dashleftarrow X_j$ are $X_i \to X_j$ and $X_i \not\!\!- X_j$. Both of them are compatible with G_C.
If restriction $X_i \not\dashleftarrow X_j \land X_i \not\to X_j$ or $X_i \not\dashleftarrow X_j \land X_i \not\dashrightarrow X_j$ is put, only $X_i \not\!\!- X_j$ is allowed and C_i, C_j are not adjacent, so $X_i \not\!\!- X_j$ is compatible with G_C. □

Strictly speaking, in Theorem 3.13 (iii), $C_i \dashrightarrow \land C_i \not\to C_j$ also implies that for any C_{k-1}, C_k on the directed paths $C_i \to ... \to C_{k-1} \to C_k \to ... \to C_j$, $dir(C_{k-1}, C_k)$ holds. For $bk(G_C)$ however, these constraints are already included due to the $dir(C_i, C_j)$ clauses. Hence this is not added in Theorem 3.13, as Equation 3.8 holds regardless.

The generalization to C-ADMGs is fairly straightforward with the addition of the constraints $X_i \leftrightarrow X_j$ and $X_i \not\leftrightarrow X_j$ to indicate presence or absence of a latent confounder. The DAG-constraints then are extended by bidirected constraints:

Theorem 3.14 (Pairwise characterization of C-ADMGs). *Let the setup be the same as in Theorem 3.13. To include bidirected constraints, the rules (i)-(iv) from 3.13 get extended by*

(v)

$$C_i \not\leftrightarrow C_j \iff \bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\leftrightarrow X_j =: nlat(C_i, C_j)$$

The background knowledge is then denoted as

$$bk(G_C) = \bigwedge_{C_i, C_j} \Big(\bigwedge_{C_i \to C_j} dir(C_i, C_j) \land \bigwedge_{\substack{C_i \dashrightarrow C_j \\ C_i \not\to C_j}} anc(C_i, C_j) \land \bigwedge_{\substack{C_i \not\dashrightarrow C_j \\ C_i \not\dashleftarrow C_j}} nrel(C_i, C_j) \land \bigwedge_{C_i \not\leftrightarrow C_j} nlat(C_i, C_j) \Big) \tag{3.9}$$

and

$$G \textit{ compatible with } G_C \iff bk(G_C) \textit{ is true in } G \tag{3.10}$$

Proof. (v) If $C_i \not\leftrightarrow C_j$, by C-ADMG definition that means for all $X_i \in C_i, X_j \in C_j$ it is $X_i \not\leftrightarrow X_j$ which exactly implies $nlat(C_i, C_j)$. For the reverse direction, if $nlat(C_i, C_j)$ it means for no $X_i \in C_i, X_j \in C_j$ it is $X_i \leftrightarrow X_j$. This means the C-ADMG does not have $C_i \leftrightarrow C_j$.
The rest follows analogously to the proof of Theorem 3.13 □

Note that $C_i \leftrightarrow C_j$ does not place a restriction on $X_i \in C_i, X_j \in C_j$: there may or may not be a latent confounder s.t. $X_i \leftrightarrow X_j$.

This shows that any C-ADMG can be described as a conjunction of pairwise constraints. The reverse direction may not be true. In general, an arbitrary set of pairwise background knowledge constraints can not be described by a C-ADMG, as the following example shows. This is because C-ADMG pairwise representation does not include required edges $X_1 \to X_2$ or required ancestral relationships $X_1 \dashrightarrow X_2$. Consider the fully connected PDAG in Figure 3.9, which entails the pairwise constraint $X_1 \to X_2$.

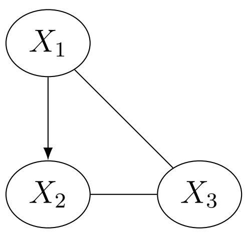

Figure 3.9.: Pairwise background knowledge irrepresentable by C-DAG.

A clustering would need $X_1 \in C_1, X_2 \in C_2$ with $C_1 \to C_2$, but there is no way to include X_3 in those clusters without introducing an additional directed edge. So there is no way to model the constraint $X_1 \to X_2$ with C-ADMGs.

Since C-ADMGs and C-DAGs can be denoted as a boolean combination of pairwise constraints (Theorems 3.13 and 3.14), the theory on representation of causal background knowledge developed in [Fan+22] can directly be transferred to C-ADMGs, analogous to [Fan+22] Appendix B. In the case of C-ADMGs, since pairwise background knowledge does not explicitly model bidirected edges (and therefore, if latent variables are present, would not put any restrictions on where they could be), the bidirected component of $bk(G_C)$ would need to be dropped for the C-ADMG analysis. It would be replaced with no assumption on bidirected edges. This is similar to Appendix B in [Fan+22], where they represent tiered background knowledge [ASC20] as a boolean combination of pairwise constraints and ignore in-tier confounding.

In conclusion, the pairwise representation of C-ADMGs embeds them formally in the mathematical language of first order logic [Bar77], with DAGs as the objects of reasoning and pairwise constraints as predicates. That allows easy comparison with other pairwise and aggregate background knowledge frameworks, as well as easier transfer of results from those other background knowledge frameworks to C-ADMGs and vice versa via the pairwise route.

3.2. Tiered background knowledge

When thinking about how to organize variables into groups for causal discovery, why not sort them by when they happened? This is the idea of tiered background knowledge ([ASC20], [BD23]). Variables are partitioned into tiers, and variables in later tiers cannot cause variables in earlier tiers, presumably because they happened earlier in time. Tiered background knowledge shares some similarities with C-ADMGs and are therefore discussed and compared with them here.

Definition 3.15 (Tiered background knowledge [ASC20]). *I say that a MAG satisfies tiered background knowledge if the variables may be partitioned into $n > 1$ disjoint subsets (tiers) $T = \{T_1, ..., T_m\}$ and for all $A \in T_i$ and $B \in T_j$ with $1 \leq i < j \leq m$ either*

(i) A is an ancestor of B or

(ii) A and B are not adjacent.

Obviously, this definition takes potential latent variables into account due to the MAG. An example showing how tiered background knowledge works is given in the graph in Figure 3.10.

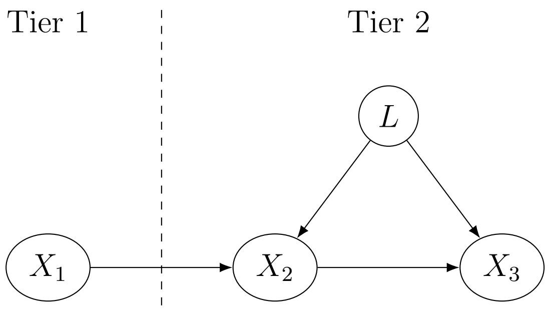

Figure 3.10.: Example of tiered background knowledge from [ASC20]

With $T_1 = \{X_1\}, T_2 = \{X_2, X_3\}$, the partition $T = \{T_1, T_2\}$ is tiered background knowledge. L is an unobserved latent confounder.

3.2.1. Pairwise characterization and comparison to C-DAGs and C-ADMGs

The pairwise notation of tiered background knowledge follows directly from Definition 3.15 and was first given in [Fan+22] Appendix B:

$$tbk(T) := \bigwedge_{\substack{X_i \in T_i, X_j \in T_j \\ 1 \leq i \leq j \leq m}} X_i \dashrightarrow X_j \vee (X_i \not\rightarrow X_j \wedge X_i \not\leftarrow X_j). \tag{3.11}$$

This is a boolean combination of pairwise constraints and [Fan+22] show in their Appendix B that their results about pairwise background knowledge can be extended to tiered background knowledge.

Equation 3.11 is different from the equations in Theorems 3.13 and 3.14, even though the two ideas of tbk and C-ADMGs seem quite similar. This section compares the differences of tbk and C-ADMGs and shows how tbk can be viewed as a C-ADMG (with a small change in C-ADMG definition).

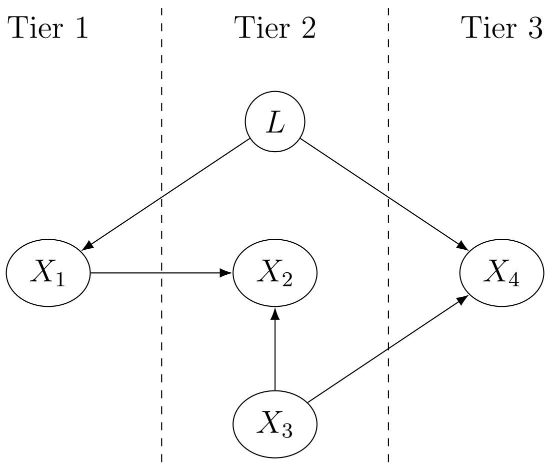

Figure 3.11.: No tiered background knowledge.

Cross-tier confounding is an issue for tiered background knowledge and cannot be modeled by it. Consider the example in Figure 3.11.

L is not assigned to any tier, as it would be projected away when producing the MAG. Only the observed variables belong to tiers. Construction of the MAG for this graph is very straightforward as no edges need to be added due to no inducing paths, so it is just the latent projection shown in Figure 3.12.

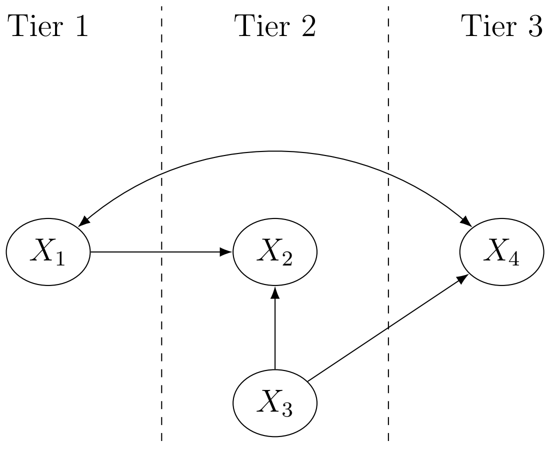

Figure 3.12.: MAG for no tiered background knowledge.

The problem is that X_1 is in an earlier tier compared to X_4, but X_1 is not an ancestor of X_4, so Definition 3.15 (i) is violated. In the MAG however it is $X_1 \leftrightarrow X_4$, so they are adjacent, therefore Definition 3.15 (ii) is also violated. This means for a situation like in Figure 3.11, it is impossible to construct tiered background knowledge with X_1, X_4 in separate tiers. The only way to handle this with tbk is to put all variables into the same tier, which effectively means no assumption is made, and the tbk does not add any knowledge.

Note that 'no cross-tier confounding' is a restricting assumption that C-ADMGs can include: the analogous clustering would result in the C-ADMG G_C in Figure 3.13 with $C = \{C_1, C_2, C_3\}$ and $C_1 = \{X_1\}, C_2 = \{X_2, X_3\}$, $C_3 = \{X_4\}$.

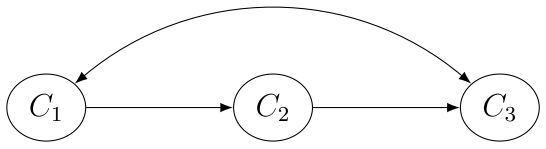

Figure 3.13.: C-ADMG when no tiered background knowledge possible.

A second advantage of C-ADMGs is their ability to rule out ancestral relationships across tiers, which tiered background knowledge cannot do. Consider the DAG in Figure 3.14.

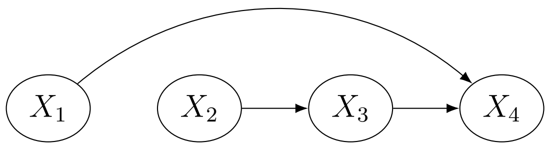

Figure 3.14.: Possibility of ruling out ancestral relationships.

Putting the nodes into tiers $T_1 = \{X_1\}, T_2 = \{X_2, X_3\}, T_3 = \{X_4\}$ would result in the assumption about X_1, X_2 (and same for X_1, X_3) being: $X_1 \dashrightarrow X_2 \vee (X_1 \not\rightarrow X_1 \wedge X_1 \not\leftarrow X_2)$. When putting nodes into clusters $C_1 = \{X_1\}, C_2 = \{X_2, X_3\}, C_3 = \{X_4\}$, the C-DAG assumption is stricter for X_1, X_2 (and also for X_1, X_3) as it implies $X_1 \not\dashrightarrow X_2 \wedge X_1 \not\dashleftarrow X_2$, which implies $X_1 \not\rightarrow X_2 \wedge X_1 \not\leftarrow X_2$ and therefore implies the tiered assumption $X_1 \dashrightarrow X_2 \vee (X_1 \not\rightarrow X_1 \wedge X_1 \not\leftarrow X_2)$. In this case, tbk cannot rule out edges $X_1 \rightarrow X_2, X_1 \rightarrow X_3$, but the C-DAG can, as it does not have an arrow between C_1 and C_2.

In fact, conditions (i) and (ii) from the C-DAG pairwise representation Theorem 3.13 alone are almost equivalent to tiered background knowledge assumptions (cf. Theorem 3.17), but (iii) and (iv) specify constraints that tiered background knowledge cannot model.

C-DAGs (not C-ADMGs) are the correct point of comparison to tbk, because tiered background knowledge does not allow for cross-tier confounding, which in an equivalent C-DAG would mean all the bidirected edges are between nodes in the same cluster, so the cluster graph would indeed be a C-DAG and not contain bidirected edges (but this C-DAG would allow confounding within clusters). Tiered background knowledge does not entail any required edges, so it makes sense to compare tbk to a slightly changed version of C-DAGs, the relaxed C-DAGs. These also do not contain any required edges.

Definition 3.16 (Relaxed C-DAG). *A relaxed C-DAG G_C is a C-DAG where $C_i \rightarrow C_j$ in pairwise characterization implies $\bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\dashleftarrow X_j$.*

This changes the pairwise characterization of a C-DAG from Theorem 3.13 (ii) in the way that $C_i \rightarrow C_j$ must not mean that there exist $X_i \in C_i, X_j \in C_j$ s.t. $X_i \rightarrow X_j$. In other words, the clause $\bigvee_{X_i \in C_i, X_j \in C_j} X_i \rightarrow X_j$ is dropped.

With this minor change in C-DAG definition, one can show that relaxed C-DAGs are more expressive than tbk:

Theorem 3.17 (Relaxed C-DAGs are more expressive than tbk (tbk $\subsetneq$ relaxed C-DAG)). *Any tiered background knowledge $T = \{T_1, ..., T_m\}$ can be represented by a relaxed C-DAG G_C with $T_i = C_i$ for all i and $C = \{C_1, ..., C_m\}$. The relaxed C-DAG corresponding to tbk is of the following form: for $1 \leq i < j \leq m$: $C_i \rightarrow C_j$.*
Tbk is a strict subset of relaxed C-DAGs: there exist relaxed C-DAGs that can not be represented by tbk without a loss of information on the constraints.

Proof. For the given relaxed C-DAG only (i) of Theorem 3.13 and
$C_i \rightarrow C_j \Rightarrow \bigwedge_{X_i \in C_i, X_j \in C_j} X_i \not\dashleftarrow X_j$ are applicable, as the constructed relaxed C-DAG does not entail the constraints $C_i \dashrightarrow C_i \wedge C_i \nrightarrow C_j$ or $C_i \not\dashrightarrow C_j \wedge C_i \not\dashleftarrow C_j \wedge C_i \neq C_j$. This is because for all clusters C_i, C_j either $C_i \rightarrow C_j$ or $C_j \rightarrow C_i$.
One needs to show that for any C_i, C_j if and only if their pairwise constraint from Theorem 3.13 is satisfied, the conditions of tbk from Definition 3.15 are satisfied. If $C_i = C_j$, neither relaxed C-DAG nor tbk make any assumption, so relaxed C-DAG constraint iff tbk constraint is satisfied. So let $C_i \rightarrow C_j$, the only other option.
It is $C_i = T_i, C_j = T_j$ and for $i \neq j$ the

- relaxed C-DAG implies: for $X_i \in C_i, X_j \in C_j$: $X_i \not\dashleftarrow X_j$.
- tbk implies: for $X_i \in C_i, X_j \in C_j$: $X_i \dashrightarrow X_j \vee (X_i \nrightarrow X_i \wedge X_i \nleftarrow X_j)$.

"$\Rightarrow$ " (relaxed C-DAG implies tbk): First consider that if $X_i \not\dashleftarrow X_j$ the options left for X_i, X_j are $X_i \rightarrow X_j \vee (X_i \dashrightarrow X_j \wedge X_i \nrightarrow X_j) \vee (X_i \not\dashrightarrow X_j \wedge X_i \not\dashleftarrow X_j)$. If $X_i \rightarrow X_j \vee (X_i \dashrightarrow X_j \wedge X_i \nrightarrow X_j)$ then X_i is an ancestor of X_j, so condition (i) of tbk is fulfilled. If $(X_i \not\dashrightarrow X_j \wedge X_i \not\dashleftarrow X_j)$ this implies $(X_i \nrightarrow X_j \wedge X_i \nleftarrow X_j)$ and therefore X_i, X_j are not adjacent, fulfilling tbk Definition 3.15 (ii).
"$\Leftarrow$ " (tbk implies relaxed C-DAG): If X_i is an ancestor of X_j this means $X_i \rightarrow X_j \vee (X_i \dashrightarrow X_j \wedge X_i \nrightarrow X_j)$, which would imply $X_i \not\dashleftarrow X_j$ by acyclicity. If X_i is not an ancestor of X_j, they are in different tiers (clusters) by construction, then X_i and X_j cannot be adjacent by tbk. Thus the only option left is $X_i \not\dashrightarrow X_j \wedge X_i \nrightarrow X_j \wedge X_i \nleftarrow X_j$ which reduces to $X_i \not\dashrightarrow X_j \wedge X_i \nleftarrow X_j$. If it was $X_i \not\dashrightarrow X_j \wedge X_i \not\dashleftarrow X_j$, the proof would be done as that would imply $X_i \not\dashleftarrow X_j$. And indeed it has to be $X_i \not\dashrightarrow X_j \wedge X_i \not\dashleftarrow X_j$, as $X_i \dashleftarrow X_j \wedge X_i \nleftarrow X_j$ would mean there is a directed path $X_i \leftarrow ... \leftarrow X_k \leftarrow X_l \leftarrow ... \leftarrow X_j$ with $X_k \in C_i, X_l \in C_j$. As X_i, X_j are in different tiers, that path has to cross tiers at some point, let that be between X_k and X_l. Then for X_k, X_l the tbk would be violated, as X_k is in an earlier tier, is not an ancestor of X_l but they are adjacent. Therefore

it has to be that $X_i \not\dashrightarrow X_j \wedge X_i \not\dashleftarrow X_j$, implying $X_i \not\dashleftarrow X_j$ and therefore satisfying the relaxed C-DAG constraint between C_i and C_j.
Strict subset: As the example in Figure 3.14 shows, there exist (relaxed) C-DAGs (e.g. $C_1 \to C_2 \leftarrow C_3$ without $C_1 \to C_3$) that can not be modeled by tbk without a loss of information. $\square$

Corollary 3.18 (**C-ADMGs are more expressive than tbk**). *C-ADMGs can represent any tiered background knowledge (with $T_i = C_i$ as in Theorem 3.17), but the reverse is not necessarily true.*

Proof. Since C-ADMGs can model everything a C-DAG can model and more, with potentially bidirected edges between clusters, C-ADMGs are also more expressive than tbk by transitivity. $\square$

As I have shown, any tiered background knowledge can be represented by a fully connected relaxed C-DAG with the clusters corresponding to the tiers. In addition, it is possible to add more flexible constraints with a C-DAG or C-ADMG, which is not possible with tbk.

This however raises the question whether an adapted FCI algorithm ([ASC20] used FCITiers with tbk) can still be applied to any C-DAG or C-ADMG, a problem that is discussed in Chapter 5. First, the FCITiers algorithm is discussed in the next Section 3.2.2.

3.2.2. Causal discovery with tiered background knowledge

[ASC20] show that the FCI algorithm is sound and complete with tiered background knowledge and give a modified FCI algorithm, called FCITiers, that takes into account the tiered background knowledge structure.

Definition 3.19 (**Edges(G,S)**). *Let S be a subset of the variables in G. $Edges(G, S)$ is the subset of edges in G connecting two members of S.*

This set will be needed in Algorithm 5.

Algorithm 5 FCITiers [ASC20]

```
Require: Disjoint tiers $T = \{T_1, ..., T_m\}$
1: form an unconnected graph $\mathcal{P}$ over $\bigcup_{i=1}^{m} T_i$
2: for i = m to 1 do
3:     $A_i \leftarrow \bigcup_{j=1}^{i-1} T_j$
4:     $B_i \leftarrow T_i$
5:     $O_i \leftarrow A_i \cup B_i$
6:     $\mathcal{P}_i = FCIExogenous(A_i, B_i)$
7:     add $Edges(\mathcal{P}_i, O_i)$ to $\mathcal{P}$
8: return PAG $\mathcal{P}$
```

Algorithm 6 FCIExogenous [ASC20]

```
Require: Disjoint sets $A$ and $B$
 1: Form an unconnected graph $\mathcal{P}$ over $A \cup B$
 2: Add $X_i \rightarrow X_j$ to $\mathcal{P}$ for all $X_i \in A$ and $X_j \in B$
 3: Add $X_i \circ\!\!-\!\!\circ X_j$ to $\mathcal{P}$ for all $X_i, X_j \in B$
 4: $d \leftarrow 0$
 5: repeat
 6:     for all pairs of adjacent vertices $(X_i, X_j) \in A \cup B$ and subsets $S \subset adj(X_i) \setminus \{X_j\}$
        s.t. $|S| = d$ do
 7:         if $X_i \perp\!\!\!\perp X_j \mid S$ then
 8:             delete edge $X_i *\!\!-\!\!* X_j$ from $\mathcal{P}$
 9:             $sepset(X_i, X_j) \leftarrow S$
10:             $sepset(X_j, X_i) \leftarrow S$
11:     $d \leftarrow d + 1$
12: until $d > |adj(X_i) \setminus \{X_j\}|$ for all pairs of adjacent vertices $(X_i, X_j) \in A \cup B$
13: apply $\mathcal{R}_0$ to $\mathcal{P}$
14: for all pairs of adjacent vertices $(X_i, X_j) \in A \cup B$ do
15:     if there exists $S \in pds(X_i, X_j)$ s.t. $X_i \perp\!\!\!\perp X_j \mid S$ then
16:         delete edge $X_i *\!\!-\!\!* X_j$ from $\mathcal{P}$
17:         $sepset(X_i, X_j) \leftarrow S$
18:         $sepset(X_j, X_i) \leftarrow S$
19: for all pairs of adjacent vertices $(X_i, X_j) \in B$ do
20:     replace edge $X_i *\!\!-\!\!* X_j$ with $X_i \circ\!\!-\!\!\circ X_j$ in $\mathcal{P}$
21: exhaustively apply $\mathcal{R}_0 - \mathcal{R}_4, \mathcal{R}_8 - \mathcal{R}_{10}$ to $\mathcal{P}$
22: return PAG $\mathcal{P}$
```

The orientation rules $\mathcal{R}_0 - \mathcal{R}_4, \mathcal{R}_8\mathcal{R}_{10}$ guarantee that the resulting PAG is maximally informative, i.e. that the algorithm is complete. They can be found in the supplement of [ASC20], and are not described here as they are not directly relevant for this thesis.

Algorithm 5 starts in the latest tier T_m and works its way up. It always divides the vertice set into the current tier $B_i = T_i$ and the preceding tiers $A_i = \bigcup_{j=1}^{i-1} T_j$. Tiers below the current tier, i.e. $T_{i+1}, ..., T_m$ are not considered, as their nodes cannot be part of connecting paths, so they do not have to be considered for separating sets. This is due to any path which includes a node X from one of these lower tiers must have a v-structure at X or somewhere in that lower tier due to the tbk assumptions (roughly speaking, high tier $\rightarrow$ low tier $\leftarrow$ high tier).

FCITiers calls FCIExogenous m times and adds the edges from the subroutine bit by bit. FCIExogenous only connects B_i among itself and A_i with B_i. A_i is not connected among itself, edges between two nodes $X, Y \in A_i$ are only considered at a later stage, when $X, Y \in B_k$ happens later for a smaller k. For each of these A_i, B_i combinations, it runs the standard FCI algorithm, without considering edges within A_i and forcing all edges between $A \in A_i, B \in B_i$ to be of form $A \rightarrow B$.

3.3. Typing assumption

The typing assumption [Bro+22] is the third and last aggregate background knowledge variant similar to C-DAGs and C-ADMGs presented in this thesis. In this framework, a CPDAG is assumed to be known and variables are grouped according to types (quite similar to clusters). If you then assume type consistency (cf. Definition 3.22), it is possible to orient additional edges in a CPDAG.

Definition 3.20 (**t-Dag** [Bro+22]). *A t-DAG with k types is a DAG $G = (V, E)$ augmented with a mapping $T : V \to \mathcal{T}$ such that the edge of $X_i \in V$ is $T(X_i) = t_j \in \mathcal{T}$, where $|T| = k$.*

The mapping fulfills a very similar role to the clusterings and tiers discussed previously, they group variables together and with the t-edge defined below, define relationships between those groups of variables.

Definition 3.21 (**t-edge** [Bro+22]). *A t-edge $E(t_i, t_j)$ is the set of edges that goes from a vertex of type t_i to a vertex of type t_j. More formally, $E(t_i, t_j) = \{(X_k, X_l) \in E \mid T(X_k) = t_i, T(X_l) = t_j\}$ for any pair of types $t_i, t_j \in \mathcal{T}$ with $t_i \neq t_j$.*

The definition of a t-edge is different to cluster edges or tier orderings. A t-edge is not the set of possible (and forbidden) edges and does not imply ancestral relationships, as was the case with clusterings or tiers. A t-edge is the collection of edges between two types.

Next comes the assumption of type consistency, which means edges between two types have to have one common direction. For example, a t-edge like
$E(t_1, t_2) = \{(X_1, X_2), (X_3, X_1)\}$ with $T(X_1) = t_1, T(X_2) = t_2, T(X_3) = t_2$ would not be consistent, as the t-edge contains edges of the form $t_1 \to t_2$ and $t_2 \to t_1$.

Definition 3.22 (**Consistent t-DAG** [Bro+22]). *A consistent t-DAG is a t-DAG where for every pair of distinct types t_i, t_j, if t-edge $E(t_i, t_j) \neq \emptyset$ one has that $E(t_i, t_j) = \emptyset$. This structural constraint is called type consistency. For conciseness, $t_i \xrightarrow{t} t_j$ denotes $E(t_i, t_j) \neq \emptyset$.*

A key difference between typing assumption and C-DAGs is that C-DAGs do not allow cycles, while the t-edges can form a cycle of length > 2. Only cycles of length two (cycles to the same type) are forbidden by type consistency.

As an example, consider the type mapping $T(X_1) = t_1, T(X_2) = t_2, T(X_3) = t_2, T(X_4) = t_3$ with DAG shown in Figure 3.15.

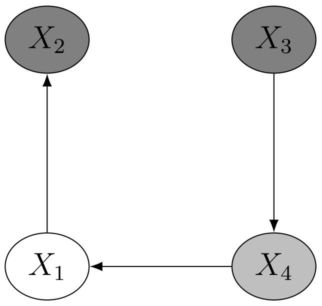

Figure 3.15.: Example of three type cycle.

All t-edges are consistent (e.g. $E(t_1, t_2) = \{(X_1, X_2)\}$ and $E(t_2, t_1) = \emptyset$), but there is a cycle of length three: $t_1 \overset{t}{\rightarrow} t_2 \overset{t}{\rightarrow} t_3 \overset{t}{\rightarrow} t_1$. [Bro+22] argue that this setup is less restricting as one does not have to specify directions between types beforehand (clusters and tiers have to do that) and leads to interesting results.

The usefulness of a consistent t-DAG is as follows: When given a CPDAG G^* together with a mapping T, if that mapping is consistent w.r.t. the true underlying DAG G, i.e. the t-DAG G_T is consistent, additional edges can be oriented.

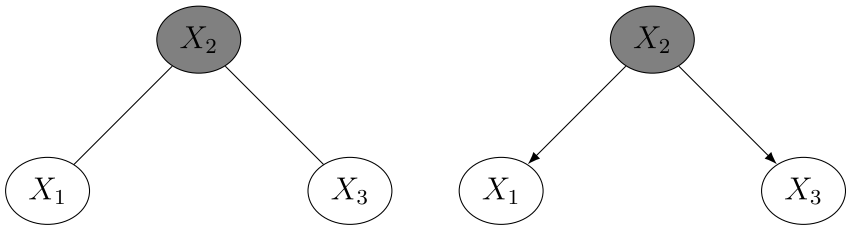

Figure 3.16.: Two-type fork: additional orientation by types $T(X_1), T(X_3) = t_1$ and $T(X_2) = t_2$.

The example in Figure 3.16 shows how. The graph on the left is the CPDAG and assume that types $T(X_1), T(X_3) = t_1$ and $T(X_2) = t_2$ are given. The Markov equivalence class without types is $X_1 \rightarrow X_2 \rightarrow X_3$, $X_1 \leftarrow X_2 \leftarrow X_3$ and $X_1 \leftarrow X_2 \rightarrow X_3$. However only the last of these DAGs is type consistent with mapping T, so both edges can be oriented. This special structure gets the name two-typed fork.

Definition 3.23 (**MEC of t-DAG** [Bro+22])**.** *The MEC of a t-DAG G_T is $M(G_T) := \{G' \mid G' \sim G_T\}$, where "$\sim$" denotes Markov equivalence.*

Definition 3.24 (**t-MEC of t-DAG** [Bro+22])**.** *The t-MEC of a consistent t-DAG G_T is $M_T(G_T) := \{G'_T \mid G'_T \overset{t}{\sim} G_T\}$ where "$\overset{t}{\sim}$" denotes Markov equivalence limited to consistent t-DAGs with the same type mapping T.*

The MEC of a t-DAG is just the MEC of the DAG, only with the addition of the DAG also being decorated with the type mapping T. The t-MEC limits the MEC to DAGs with the same type mapping, so DAGs with edges inconsistent to the t-edges of G_T are not included.

Definition 3.25 (**Essential graph** [Bro+22]). *The essential graph G^* associated to the consistent t-DAG G_T is*

$$G^* := \bigcup_{D \in M(G_T)} G. \tag{3.12}$$

Definition 3.26 (**t-essential graph** [Bro+22]). *The t-essential graph G_T^* associated to the consistent t-DAG G_T is*

$$G_T^* := \bigcup_{G \in M_T(G_T)} G. \tag{3.13}$$

The essential graph G^* is the union of all Markov equivalent DAGs and the t-essential graph G_T^* is the union of all t-Markov equivalent DAGs. Union of edges here means that if edges $X_i \to X_j$ ($\triangleq (X_i, X_j)$) and $X_j \to X_i$ ($\triangleq (X_j, X_i)$) are merged, they become the undirected edge $X_i - X_j$ ($\triangleq ((X_i, X_j), (X_j, X_i))$).

The purpose of the t-essential graph is to graphically characterize the t-MEC of a t-DAG, similarly to how CPDAGs (or essential graphs) characterize the MEC of a DAG. The goal with the added type mapping is to reduce the size of the t-MEC compared to the size of the MEC. T-essential graph properties and size reduction of the t-MEC are beyond the scope of this thesis. They are discussed in Section 4.3. of [Bro+22].

3.3.1. Pairwise characterization and comparison to C-DAGs and C-ADMGs

As for other aggregate background knowledge frameworks, it is useful to denote them as booleans of pairwise background knowledge to compare them amongst each other.

Definition 3.27 (**Pairwise characterization of typing assumption**). *Let T be a type mapping with types $\{t_1, ..., t_k\}$. Then the restrictions T places on a DAG can be characterized as*

$$tpa(T) := \bigwedge_{t_i < t_j} \Big(\bigwedge_{\substack{T(X_i)=t_i \\ T(X_j)=t_j}} X_i \to X_j \vee X_i \not\sim X_j \Big) \otimes \Big(\bigwedge_{\substack{T(X_i)=t_i \\ T(X_j)=t_j}} X_i \leftarrow X_j \vee X_i \not\sim X_j \Big). \tag{3.14}$$

In other words, the edges between types t_i, t_j are either $\to$ or do not exist, or the edges between t_i, t_j are either $\leftarrow$ or do not exist.

In addition to the previously discussed possibility of t-cycles, Definition 3.27 shows the other substantial difference of the typing assumption compared to C-DAGs and tbk clearly: the orientation of a t-edge is not decided a priori. While the clauses in Definition 3.27 look similar to Equation 3.11 (with $X_i \to X_j$ instead of $X_i \dashrightarrow X_j$), the big difference is the $\otimes$. Direction between types is not decided, the only assumption is that all edges within a t-edge point in the same direction (or are absent).

That is a qualitatively different assumption and useful if groups of variables are known, but it is unclear in which direction causation flows between them. If direction between groups is quite clear, the typing assumption is too loose in its constraints and cannot model them restrictively enough. When comparing typing assumption to C-DAGs and C-ADMGs types correspond to clusters. While the grouping of variables is similar from types to clusters, the typing assumption does not impose explicit restrictions on the edges between types, only the implicit restriction that edges across types point in the same direction. Clusters, in comparison, put an explicit constraint like $C_i \rightarrow C_j$ or $C_i \not\rightarrow C_j$ on the edges of their members, i.e. they rule out certain edges a priori.

Additionally in comparison with C-DAGs and C-ADMGs, t-DAGs do not include bidirected edges and it is not clear how they could be incorporated into t-DAGs. If bidirected edges were allowed, they would disrupt t-edge orientation, as two nodes could be bidirected adjacent without a directed edge between them. This would make the notion of consistent t-edges unclear. Whether latent variables are a useful addition to t-DAGs and how to add them could be a topic for future research and is not discussed here.

3.3.2. Causal discovery with typing assumption

One result of [Bro+22] is that $G^*_T \subset G^*$ and edge orientations can be gained by 'synchronizing' edges across types, like in the example of Figure 3.16.

Therefore to enhance causal discovery, one uses the combination of the CPDAG (from a discovery algorithm like the PC algorithm) and the t-DAG to orient additional edges via t-propagation.

Algorithm 7 t-Propagation (G, T) [Bro+22]

```
1: Enforce type consistency: If there exists an oriented edge between pairs of variables
   with types $t_i, t_j \in \mathcal{T}$ in $G$, assume $t_i \overset{t}{\rightarrow} t_j$ and orient all edges between these types.
2: Apply the Meek orientation rules from figure 2.7 to propagate the edge orientations
   derived in step (1).
3: Repeat from step (1) until the graph is unchanged
4: Enumerate all t-DAGs that can be produced by orienting edges in the resulting
   graph. Reject any inconsistent t-DAGs and take the union of all remaining t-DAGs
   to obtain the t-essential graph.
```

If G has the same skeleton, v-structures and type mapping T as the t-essential graph G^*_T, then Algorithm 3.3.2 is guaranteed to recover the corresponding t-essential graph $M_T(G_T)$ [Bro+22].

One problem however is that with finite sample sizes, the CPDAG from the discovery algorithm may violate type consistency and cause t-propagation to fail. Another issue is that t-propagation has non-polynomial time complexity. This is due to the enumeration required in step (4), which is not skippable as the algorithm would no longer be complete. To illustrate why that is, the example given in Figure 3.16 would not be recovered as the CPDAG on the left does not contain any directed edges which could be used for Meek's orientation rules. Therefore no edges would be directed and t-propagation would not be able to synchronize edge orientations across types, as no orientations are given. Thus the enumeration in step (4) of Algorithm 3.3.2 is required for completeness.

A second approach to causal discovery presented by [Bro+22] is a typed variant of the PC algorithm. The skeleton recovery phase of PC is left untouched, but when orienting v-structures as well as edge orientations by Meek's rules, instead of orienting single edges, whole t-edges should be oriented. This means Meek's edge orientation phase is replaced with t-propagation. Regarding the v-structure orientation phase, there are two strategies leading to two different typed PC (TPC) algorithms. In addition to v-structures, two-typed forks (Figure 3.16) can also be oriented. In TPC-naive, the first encountered structure (v-structure or two-typed fork) is used to orient each t-edge. In TPC-majority, every structure that would lead to an orientation is considered, and the more prominent orientation based on all of these is used. The latter strategy is less sensitive to erroneous edge deletions or orientations.

Both of these strategies use typing assumption to enhance the output of a discovery algorithm, but do not improve the discovery algorithm itself, which, in comparison, FCITiers does. Improvement of the causal discovery algorithm is also the goal for causal discovery with C-DAGs and C-ADMGs, discussed in Chapters 4 and 5.

3.4. Conclusion and discussion

In this chapter I introduced the concept of background knowledge as well as its popular representations in current literature in both pairwise and aggregate form. I formally defined C-DAGs and C-ADMGs, which are a form of aggregate background knowledge and compared them with similar ideas like tiered background knowledge and typing assumption. Characterizing these aggregate background knowledge frameworks in terms of pairwise background knowledge allowed for examination of their differences.

A relevant question could be which version of background knowledge fits a specific case best. It depends on how much background knowledge is accessible and the confidence in said background knowledge. If background knowledge is flawed, results could be unpredictably bad and to my knowledge no research exists which tries to quantify effects of erroneous background knowledge. The least restrictive form is pairwise background knowledge, as that allows very small constraints and localized constraints. Aggregate background knowledge always forces one to sort *all* variables into groups. The least restrictive aggregate background knowledge framework is the typing assumption, since it allows for type-cycles and only orients edges after skeleton discovery. Typing assumption may however be limited, as it cannot model latent variables. In addition, if the background knowledge is good enough, typing assumption might not be restrictive enough.

For the case of latent variables, if a temporal ordering is quite clear and no cross-group confounding can be assumed, tiered background knowledge is most suitable. If additionally certain ancestral relationships can be ruled out, a C-DAG similar to tbk (as constructed in Theorem 3.17) with some edges across clusters removed could be considered. Lastly, if one would like to also model latent variables between groups, C-ADMGs are to be used. The research question may directly be inferrable from the C-ADMG via do-calculus by [Ana+23]). If that is not the case, the C-PC (Cluster-PC) or C-FCI (Cluster-FCI) from the following Chapters 4 and 5 can find application. FCITiers, C-PC and C-FCI all come with the (potential) benefit of reducing reliance on conditional independence tests, as they also directly guide the skeleton discovery phase of the constraint based methods.

For tiered background knowledge and typing assumption I already presented in this chapter how they are used to aid causal discovery. In the previous paragraph, I mentioned C-PC and C-FCI, which is a constraint based causal discovery technique using C-DAGs and C-ADMGs. These novel contributions from this thesis finally will be discussed in the next Chapters 4 and 5.

4. C-DAGs for the PC algorithm

C-DAGs were shown to possess desirable properties in direct use for causal inference by [Ana+23]. The previous chapter showed that they also share several similarities with other forms of aggregate background knowledge like tiered background knowledge and typing assumption. Those two have been used for causal discovery, so a sensible question to ask is whether C-DAGs are also suitable for causal discovery. Since C-DAGs were shown to offer more flexibility than tiered background knowledge but still retain useful topological structure unlike typing assumption, C-DAGs have a unique advantage as an aggregate background knowledge framework for causal discovery.

This chapter investigates how C-DAGs can be used for constraint based causal discovery when assuming causal sufficiency, hence no latent confounders are allowed. In particular, I adapt the PC algorithm [Spi+00] to integrate prior C-DAG knowledge, using their properties to achieve speed and accuracy improvements over the standard PC algorithm. In Section 4.1 I present how the C-DAG leads to an intitial graph, an MPDAG (cf. Definition 4.2. This graph is used as the starting graph for the Cluster-PC algorithm (cf. Algorithm 9), as opposed to the fully connected undirected graph which is used for the PC algorithm. Section 4.2 formally defines the Cluster-PC algorithm. Section 4.3 gives some examples to understand the inner workings of Cluster-PC in Section 4.3.1 and Section 4.3.2 demonstrates the results of simulation studies that empirically compare the performance of Cluster-PC to PC. Afterwards, Chapter 5 deals with absence of causal sufficency and works on adapting the FCI algorithm ([Spi+00] [Col+12a]) to C-ADMGs.

4.1. C-DAG to initial graph

How can a C-DAG help the PC algorithm? The straightforward idea is to use the edge directions from the C-DAG to orient additional edges in the CPDAG resulting from PC, similar to t-propagation 3.3.2 [Bro+22]. Missing edges from the C-DAGs can also be pruned, but if the assumptions of the PC algorithm are met, in particular causal sufficiency, and the conditional independence tests make no mistakes, all of those edges missing from the C-DAG should also have been deleted in the PC algorithm.

The approach of orienting and pruning after the PC algorithm would work, but I would like to go one step further. Instead of only combining causal discovery and background knowledge to improve the estimate, I want to *improve* causal discovery by incorporating

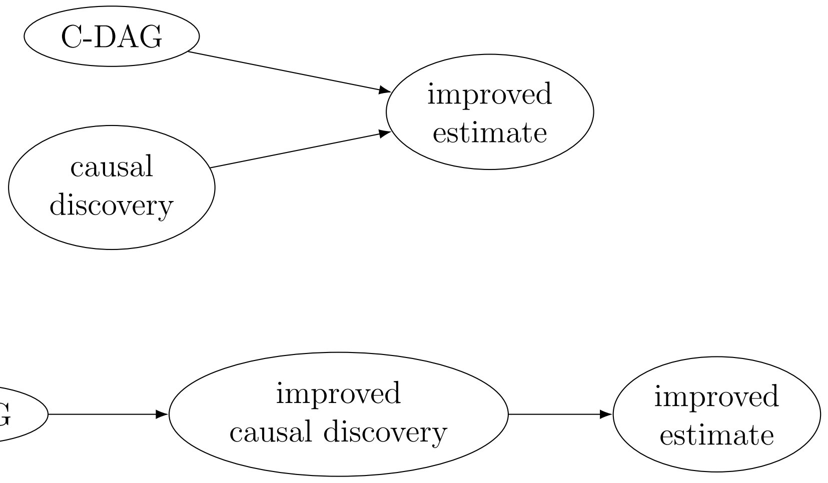

Figure 4.1.: Approaches 1 and 2 to improve causal discovery with C-DAGs.

the background knowledge directly into the discovery algorithm, as visualized in Figure 4.1.

- **Approach 1**: Use C-DAG to improve graph estimated by PC algorithm.
- **Approach 2**: Use C-DAG to improve PC algorithm.

The goal is to show that Approach 2 works at least as well as Approach 1. The hope is that Approach 2 may offer further improvements. First one has to show how Approach 1 works theoretically.

[BD23] show that for tiered background knowledge (without latent confounding within tiers), the tiered background knowledge together with a CPDAG becomes an MPDAG. The same will be shown for C-DAGs, a CPDAG together with edge orientations from a C-DAG also results in an MPDAG.

A CPDAG represents a Markov equivalence class, and orienting additional edges translates to restricting that equivalence class. In Figure 4.2, the Markov equivalence class is all fully connected directed graphs and the restricted Markov equivalence class reduces it to DAGs that satisfy $X_1 \to X_2$.

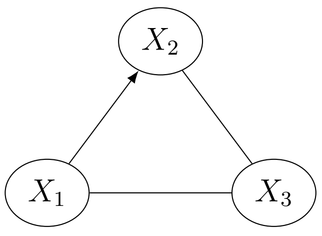

Figure 4.2.: Example for an MPDAG not representing a Markov equivalence class, but instead a restricted Markov equivalence class.

Definition 4.1 (**Restricted Markov equivalence class** [Fan+22])**.** *The restricted markov equivalence class induced by a CPDAG G^* and a pairwise causal constraint set B over V, denoted by $[G^*, B]$ is composed of all equivalent DAGs in $M(G^*)$ that satisfy B ($M(G^*)$ is the Markov equivalence class of G^*).*

A PDAG is called maximal if it is closed under Meek's orientation rules from Figure 2.7.

Definition 4.2 (**Maximally partially directed acyclic graph (MPDAG)** [Fan+22])**.** *The MPDAG H of a non-empty restricted Markov equivalence class $[G^*, B]$, induced by a CPDAG G^* and a pairwise causal constraint set B is a PDAG such that*

(i) H has the same skeleton and v-structures as G^* and

(ii) an edge is directed in H if and only if it appears in all DAGs in $[G^*, B]$.

So for a given CPDAG, the C-DAG pairwise constraint set would be:

Definition 4.3 (**C-DAG pairwise constraint set for CPDAG**)**.** *Let G^* be the CPDAG representing the Markov equivalence class $M(G)$ of G. Furthermore let G_C be a C-DAG such that G is compatible with G_C. Then the pairwise constraint set B induced by G_C is*

$$B(G_C) := \{X_i \to X_j \mid \exists C_i, C_j \in C \text{ s.t. } X_i \in C_i, X_j \in C_j, C_i \to C_j\}. \tag{4.1}$$

So $[G^*, B(G_C)]$ would result in an MPDAG H. This shows the validity of Approach 1. If one however wants to pursue Approach 2, can that same H be achieved by using a modified PC algorithm?

Recall that a C-DAG is essentially an equivalence class of DAGs, all DAGs that are compatible with G_C. And PC starts from something that can be interpreted as an equivalence class: the PC algorithm starts from a fully connected graph, which means that at start any subgraph of the fully connected graph is possible. This way, a supergraph can be interpreted as an equivalence class of graphs, too.

When PC finds independencies or orients edges, the current graph G and therefore the equivalence class is shrunk, some options for edges are taken away. The C-DAG does something very similar: when two clusters C_1, C_2 are not adjacent, their members also cannot be adjacent. And when $C_1 \to C_2$, for $X_1 \in C_1, X_2 \in C_2$, $X_1 \leftarrow X_2$ is forbidden, only $X_1 \to X_2$ or $X_1 \not\sim X_2$ (non-adjacency) are left as possible options. So C-DAGs also shrink the space of possible graphs, just like the PC algorithm.

This means one can imagine C-DAGs as a 'phase zero' in PC, before the skeleton discovery, where edges between members of non-adjacent clusters get deleted and edges between adjacent clusters get oriented according to the cluster edge. This is powered by Corollary 3.6 and does the same work as the PC algorithm, but without the need for CI tests and orientation rules. It remains to show that a PC algorithm modified like this can give the same results as Approach 1.

First one has to define the graph from 'phase zero':

Definition 4.4 (**C-MPDAG**). *The Cluster-MPDAG (C-MPDAG) H_C associated to C-DAG G_C, $C = \{C_1, ..., C_m\}, V = \bigcup_i C_i$, over nodes V is constructed by forming the unconnected graph H_C over V and by adding edges according to*

- *if $X_i, X_j \in C_k$ then add $X_i - X_j$ to G,*
- *if $X_i \in C_i, X_j \in C_j$ and $C_i \to C_j$ then add $X_i \to X_j$ to G.*

Theorem 4.5 (**C-MPDAG contains compatible graphs**). *For any G compatible with G_C, the C-MPDAG H_C associated to G_C is a supergraph of G.*

Proof. Let G be compatible with G_C. It suffices to show that any edge of G is contained in H_C. Let $X_i, X_j \in V$ and w.l.o.g. let $X_i \to X_j$ in G. If $X_i, X_j \in C_k$ in the same cluster, $(X_i, X_j), (X_j, X_i) \in E_{H_C}$ and therefore $(X_i, X_j) = X_i \to X_j \in E_{H_C}$. If $X_i \in C_i, X_j \in C_j, C_i \neq C_j$ and C_i, C_j adjacent, since $X_i \to X_j$ it is $C_i \to C_j$ and therefore $X_i \to X_j \in E_{H_C}$. Lastly $X_i \in C_i, X_j \in C_j, C_i \neq C_j$ and C_i, C_j not adjacent is not possible, as $X_i \to X_j$ would violate compatibility of G with G_C. □

The C-MPDAG is not generally a supergraph of the CPDAG G^*, as the example $X_1 \to X_2, X_1 \in C_1, X_2 \in C_2$ shows. The C-MPDAG H_C would be $X_1 \to X_2$, but G^* would be $X_1 - X_2$. The skeleton of H_C however is guaranteed to be a supergraph of G^*.

This leads to a modified PC algorithm:

The next theorem shows that the modified PC algorithm in Algorithm 8 produces the same result.

Theorem 4.6 (**Equivalence of Approaches 1 and 2**). *The modified PC algorithm in Algorithm 8 with C-DAG G_C will return the MPDAG $[G^*, B(G_C)]$.*

Proof. Both approaches will result in the same skeleton. If in Approach 1 the edge $X_1 - X_2$ is removed, there is a separating set S s.t. $X_1 \perp\!\!\!\perp X_2 \mid S$. In Approach 2, either the edge $X_1 - X_2$ is removed due to the nodes being in non-adjacent clusters or if they are in adjacent or the same cluster, the test $X_1 \perp\!\!\!\perp X_2 \mid S$ would have been performed eventually.
Both approaches will also result in the same edge orientations. If in Approach 1 the edge between nodes X_1, X_2 is oriented to $X_1 \to X_2$ due to collider or orientation rules in PC, the same happens in Approach 2. If the edge is oriented to $X_1 \to X_2$ in Approach 1 due to the nodes being in adjacent clusters, the edge would also be oriented in the skeleton pruning phase of Algorithm 8. Lastly, if the edge is oriented $X_1 \to X_2$ in Approach 1 due to orientation rules resulting from C-DAG edge orientations $B(G_C)$, the same orientation rules would also orient $X_1 \to X_2$ in Algorithm 8. □

If Approaches 1 and 2 result in the same output, why consider Approach 2 and construct a new algorithm? The hope is that with edges pruned and oriented by prior knowledge,

Algorithm 8 Modified PC algorithm

```
Require: Joint distribution $P_X$ of $d$ variables, independence oracle, C-DAG $G_C$ compatible with ground truth graph
1: $V \leftarrow \{X_1, \dots, X_d\}$, $E \leftarrow \{(X_i, X_j) \in V^2 | i \neq j\}$   ▷ form fully connected undir. graph
2: for $C_i, C_j \in C$ do   ▷ prune skeleton via C-DAG
3:     if $C_i \rightarrow C_j$ then
4:         for $X_i \in C_i, X_j \in C_j$ do
5:             $E \leftarrow E \setminus \{(X_j, X_i)\}$
6:     if $C_i, C_j$ not adjacent in $G_C$ then
7:         for $X_i \in C_i, X_j \in C_j$ do
8:             $E \leftarrow E \setminus \{(X_i, X_j), (X_j, X_i)\}$
9: for $k = 0, \dots, d-2$ do
10:     for each adjacent pair $X_i, X_j \in V$ do
11:         for all $S \subset V \setminus \{X_i, X_j\}$ with $|S| = k$ (and $S - X_i$ or $S - X_j$) do
12:             if $X_i \perp\!\!\!\perp X_j | S$ then
13:                 $E \leftarrow E \setminus \{(X_i, X_j), (X_j, X_i)\}$   ▷ remove $X_i - X_j$
14:                 $S_{i,j} \leftarrow S$
15: for each triple $X_i, X_j, X_k \in V$ with $X_i - X_j - X_k$, $X_i \leftarrow X_j - X_k$ or $X_i \rightarrow X_j - X_k$
    and $X_i \not- X_k$ do   ▷ find v-structures
16:     if $X_j \notin S_{i,k}$ then
17:         $E \leftarrow E \setminus \{(X_j, X_i), (X_j, X_k)\}$   ▷ orient the edges as $X_i \rightarrow X_j \leftarrow X_k$
18: successively apply Meek's orientation rules from Figure 2.7
19: return CPDAG $G = (V, E)$
```

the skeleton pruning phase is able to avoid CI tests that may have been faulty and resulted in (potentially) cascading errors. In addition, the edges removed in the C-DAG do not have to be CI tested. The bigger the separating set for an edge is, the more CI tests can be avoided with this and the algorithm finishes quicker. This is especially useful when the CI test of choice is expensive to compute. In total, with Approach 2 one can expect a faster algorithm and a better resulting graph.

4.2. Cluster PC algorithm

This section takes the modified PC algorithm from Algorithm 8 and adds some improvements to transform it into the Cluster-PC algorithm (C-PC).

As discussed in Sections 2.3.1 and 2.3.3, the order in which CI tests are performed matters for speed. C-DAGs with their topological structure make an additional optimization possible. Recall that for non-adjacent nodes X_1, X_2 a separating set is in either pa_{X_1} or pa_{X_2}. If in the C-DAG $C_1 \rightarrow C_2$ and $X_1 \in C_1, X_2 \in C_2, C_1 \rightarrow C_2$ then $X_2 \notin an_{X_1}$ and from Proposition 2.27 it follows that $mns_{X_2}(X_1) \subset pa_{X_2}$ (recall that $mns_{X_2}(X_1)$ is the minimal neighbor separator, cf. Definition 2.26, which for X_1, X_2 is the smallest set in the neighbors of X_2 s.t. $X_1 \perp\!\!\!\perp X_2 \mid mns_{X_2}(X_1)$). This means it suffices to search in adj_{X_2} for a separating set. One can reduce this even further: if $X_3 \in adj_{X_2}$, either $X_3 \rightarrow X_2$ when X_3 is in a parent cluster, $X_3 - X_2$ because X_2, X_3 are in the same cluster or $X_2 \rightarrow X_3$ when X_3 is in a child cluster. If X_3 is in a child cluster, it is also $X_3 \notin an_{X_1}$ and any path between X_1, X_2 going through X_3 would include a collider and therefore be blocked by default. Those paths do not need to be tested for. This means for adjacent $X_1 \rightarrow X_2$ one only needs to search for a separating set in the non-childs of X_2: $nch_{X_2} = adj_{X_2} \setminus ch_{X_2}$. For adjacent $X_1 - X_2$ it is enough to search in nch_{X_1} and nch_{X_2} instead of adj_{X_1} and adj_{X_2}. Note that these relationships refer to the current graph during the algorithm and not the ground truth graph.

Definition 4.7 (Non-child). *In a PDAG G for a node X the set of non-childs is the set of adjacent nodes that are either parents or connected via an undirected edge:* $nch_X^G := adj_X^G \setminus ch_X^G$.

It is now possible to state the C-PC algorithm. I present a stable version were edges are deleted only after a depth phase is completed, similar to PC-stable from [Zha+21a].

The algorithm first deletes all edges between nodes from non-adjacent clusters and orients edges between nodes from adjacent clusters in the C-DAG pruning and orienting phase. Afterwards, it works along the topological ordering of the clusters, and in each cluster phase it searches for CI tests, gradually increasing the cardinality of potential separating sets by one for each depth phase, just like the standard PC algorithm. Edges are deleted only at the end of each depth phase to ensure that the output is independent of the variable ordering [Zha+21a].

It is possible to work along the topological ordering due to the following reason: if

Algorithm 9 Cluster-PC algorithm

```
Require: Joint distribution $P_X$ of $d$ variables, independence oracle, C-DAG $G_C = (V_C, E_C), C = \{C_1, ..., C_r\}$ with clusters in topological ordering compatible with ground truth graph
 1: $V \leftarrow \{X_1, \dots, X_d\}$, $E \leftarrow \{(X_i, X_j) \in V^2 | i \neq j\}$   ▷ form fully connected undir. graph
 2: $G = (V, E)$   ▷ pa, ch, an, de, nb, sib and nch refer to this current G
 3: for $C_i, C_j \in C$ do   ▷ prune skeleton via C-DAG
 4:     if $C_i \to C_j$ then
 5:         for $X_i \in C_i, X_j \in C_j$ do
 6:             $E \leftarrow E \setminus \{(X_j, X_i)\}$   ▷ remove edge $X_i \leftarrow X_j$ and keep $X_i \to X_j$
 7:     if $C_i, C_j$ not adjacent in $G_C$ then
 8:         for $X_i \in C_i, X_j \in C_j$ do
 9:             $E \leftarrow E \setminus \{(X_i, X_j), (X_j, X_i)\}$   ▷ remove edge $X_i - X_j$
10: for $m \in [r]$ set $L_m := C_m \cup \bigcup_{C_s \in pa_{C_m}^{G_C}} C_s$ do   ▷ cluster phase for $C_m$
11:     for $k = 0, ..., |L_m| - 2$ do   ▷ with local graph $L_m$
12:         $del_E = \emptyset$   ▷ set for edge deletions after depth phase
13:         for each adjacent $X_j \in C_m$, $X_i \in pa_{X_j}$ do
14:             for all $S \subset nch_{X_j} \setminus \{X_i\}$ and $|S| = k$ do
15:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
16:                     $del_E \leftarrow (X_i, X_j)$   ▷ $(X_j, X_i)$ already removed by C-DAG
17:                     $S_{i,j} \leftarrow S$
18:         for each adjacent $X_i, X_j \in C_m$ do
19:             for all $S \subset nch_{X_j} \setminus \{X_i\}$ or $S \subset nch_i \setminus \{X_j\}$ and $|S| = k$ do
20:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
21:                     $del_E \leftarrow (X_i, X_j), (X_j, X_i)$
22:                     $S_{i,j} \leftarrow S$
23:         delete edges from $del_E$ in $G$
24: for each triple $X_i, X_j, X_k \in V$ with $X_i - X_j - X_k$, $X_i \leftarrow X_j - X_k$ or $X_i \to X_j - X_k$
    and $X_i \not\sim X_k$ do   ▷ find v-structures
25:     if $X_j \notin S_{i,k}$ then
26:         $E \leftarrow E \setminus \{(X_j, X_i), (X_j, X_k)\}$   ▷ orient the edges as $X_i \to X_j \leftarrow X_k$
27: successively apply Meek's edge orientation rules from Figure 2.7
28: return MPDAG $G = (V, E)$
```

$X_i, X_j \in C_m$ are in the same cluster and they are d-separable, the set S separating them must be in $L_m := C_m \cup \bigcup_{C_s \in pa^{G_C}_{C_m}} C_s$. The set L_m is called the local nodes or local graph for C_m. If $X_i \in C_i, X_j \in C_j, C_i \to C_j$, the set d-separating them must be in pa_{X_i} and therefore also in L_m.

One might think a problem arises for the orientation due to the algorithm removing some edges but not adding separating sets in the C-DAG pruning stage. Maybe some v-structures can not be oriented then? Fortunately this can not occur.

Theorem 4.8 (**No problems with separating sets**). *The C-PC algorithm in Algorithm 9 does not add separating sets for removed edges in the C-DAG pruning phase. This does not lead to wrong v-structure orientations.*

Proof. A potential problem arises if there is a triple (X_i, X_j, X_k), X_i, X_k not adjacent and no CI test like $X_i \perp\!\!\!\perp X_k \mid S$ with $X_j \in S$ were performed in C-PC because that edge was revmoved in the C-DAG pruning phase. In that case C-PC would orient $X_i \to X_j \leftarrow X_k$ as $X_j \notin S_{i,k} = \emptyset$. That could however be false as the example $X_i \to X_j \to X_k$ with X_i, X_j, X_k all in different clusters shows. I will now show that this situation can not arise.
If there is a triple (X_i, X_j, X_k), X_i, X_k not adjacent and no CI test like $X_i \perp\!\!\!\perp X_k \mid S$ with $X_j \in S$ were performed in C-PC, then X_i, X_j, X_k must be all in different clusters. That X_i, X_k are in different clusters is clear due to the edge being removed in the C-DAG pruning phase. X_j being adjacent to X_i (X_k) means it either is in a cluster adjacent to $X_i(X_k)$ or in the same cluster with X_i (X_k). But in the latter case, X_j being in the same cluster as X_i (X_k) would mean X_i and X_k would also be in adjacent clusters, in contradiction to their edge being removed in the C-DAG pruning phase. So X_i, X_j, X_k are all in separate clusters. This means however that during the C-DAG pruning and orientation phase, their triple already was oriented according to those different clusters. So there exist no unoriented triples that have no corresponding CI tests to decide collider orientation. □

Furthermore C-PC is complete as a consequence of Theorem 4.6.

Theorem 4.9 (**Soundness and completeness of C-PC**). *When the C-DAG G_C is compatible with the ground truth DAG G, the C-PC algorithm as stated in Algorithm 9 is sound and complete. It is sound in the sense that an edge is in its output G' if and only if it is in the ground truth DAG G and complete in the sense that it is closed under collider and Meek's orientation rules (cf. Figure 2.7).*

Proof. The C-PC algorithm in Algorithm 9 is a more efficient version of the modified PC algorithm in Algorithm 8. As the C-DAG G_C is assumed to be compatible with G, both PC and C-PC will recover the same skeleton. Furthermore Theorem 4.6 shows that any edge directed by adding C-DAG orientations to the PC algorithm output will also be oriented in the C-PC algorithm. □

4.3. Experiments and Simulations

This chapter gives three examples on how C-PC compares to PC for small DAGs and presents the results of two large simulation studies that compare the two algorithms.

4.3.1. Examples for C-PC

This chapter is accompanied by the notebook "Cluster_PC_examples.ipynb" in the clustercausal package. There one can find the code corresponding to the following examples as well as learn how to run and evaluate C-PC on graphs which can be handcrafted or simulated. The variable naming changes to $X_0, X_1, ...$ to make working with python array's easier in the corresponding notebook.

Example 1: 3-node graph with 2 clusters

Let me begin with the simple three node chain shown in Figure 4.3.

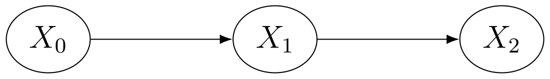

Figure 4.3.: Three node DAG example.

The Markov equivalence class and also result of the PC algorithm is the CPDAG in Figure 4.4.

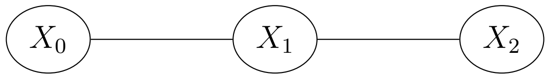

Figure 4.4.: Resulting CPDAG from PC for three node example.

Unfortunately in this case the flow of causation is ambiguous. If one was able to postulate that X_0 comes before X_1, X_2, one could establish the C-DAG $C_1 \to C_2$ with $C_1 = \{X_0\}, C_2 = \{X_1, X_2\}$. With this added information, the C-PC algorithm would start from the MPDAG in Figure 4.5.

From there, it will be able to remove the edge $X_0 \to X_2$ by CI test $X_0 \perp\!\!\!\perp X_2 \mid X_1$. Since $X_1 \in S_{0,2}$, the triple $X_0 \to X_1 - X_2$ is not oriented as a collider and therefore C-PC is able to restrict the Markov equivalence class to the true DAG from Figure 4.3. In this case, C-PC is able to recover the DAG and be much more informative than the PC algorithm.

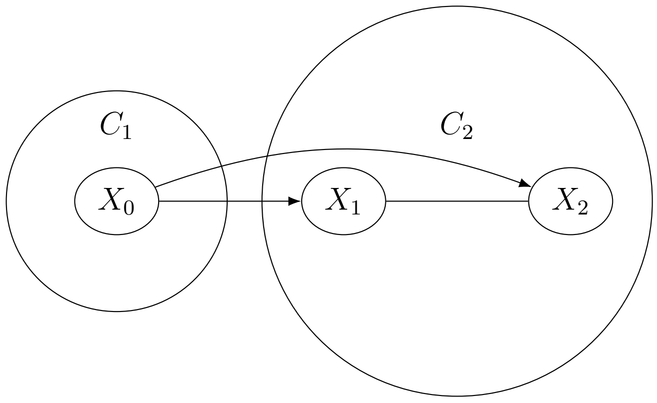

Figure 4.5.: Starting MPDAG for three node C-PC example.

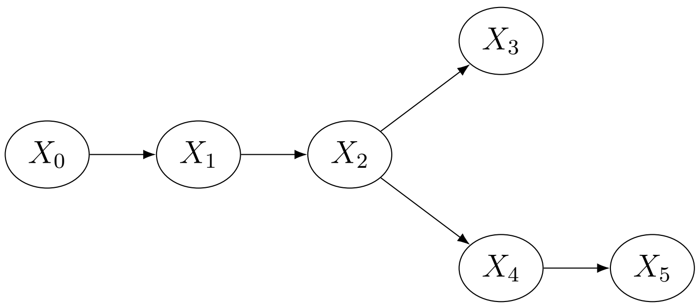

Figure 4.6.: Six node DAG example.

Example 2: 6-node graph with 3 clusters

For the second example consider the DAG in Figure 4.6.

As this graph has no v-structures, PC will not be able to orient any edge and the resulting CPDAG will be the skeleton.

This is a good example to examine how C-PC works in more detail. With clusters C_1, C_2, C_3 and $C_1 = \{X_0, X_1\}, C_2 = \{X_2, X_3\}, C_3 = \{X_4, X_5\}$ it starts from the MPDAG in Figure 4.7.

In the first phase, nodes of cluster C_1 are considered and possible d-separating sets are also contained in C_1. This means only independence $X_0 \perp\!\!\!\perp X_1$ is tested, which returns false so the edge is kept. Since all possible CI tests were performed, C-PC moves on to cluster C_2. Here nodes X_2, X_3 are considered with the local graph containing nodes $\{X_0, X_1, X_2, X_3\}$, which is the set where C-PC searches for separating sets. See the difference to PC here, X_4, X_5 are neighbors of X_2, X_3, but are not considered as possible separating nodes due to them being in a lower cluster.

For each depth level, edges between clusters are considered first and then edges within clusters. At depth 0 no edge between clusters is deleted. Also no edge within clusters, namely $X_2 - X_3$ is deleted. At depth 1 the CI tests find conditional independencies

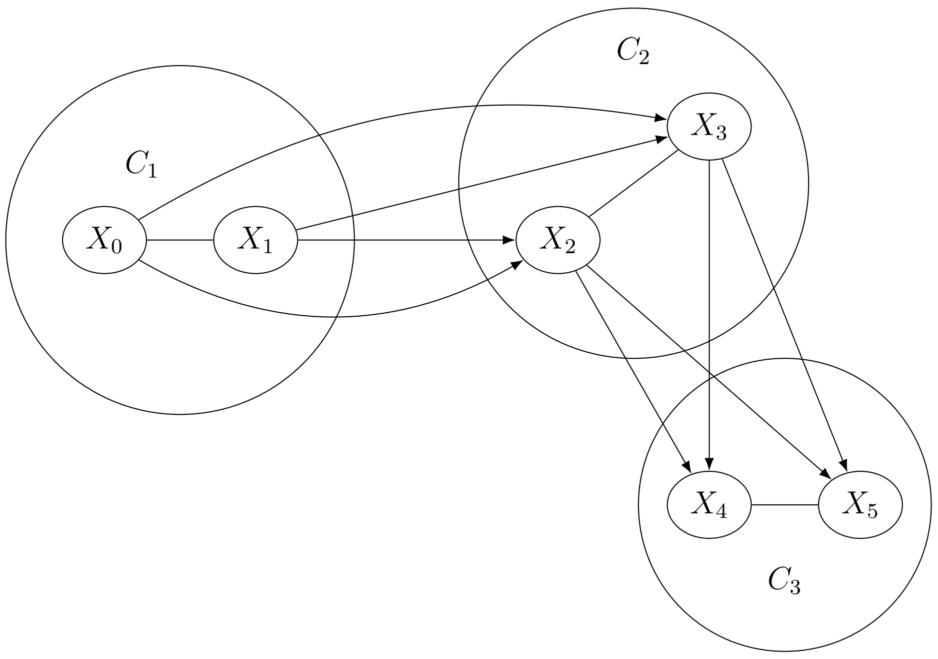

Figure 4.7.: Starting MPDAG for C-PC for six node example.

$X_0 \perp\!\!\!\perp X_2 \mid X_1$, $X_0 \perp\!\!\!\perp X_3 \mid X_1$, $X_0 \perp\!\!\!\perp X_3 \mid X_2$ and $X_1 \perp\!\!\!\perp X_3 \mid X_2$, so those edges are deleted and $S_{0,2} = \{X_1\}, S_{0,3} = \{X_1, X_2\}$ and $S_{1,3} = \{X_2\}$. Edge $X_2 - X_3$ stays. Now C_2 phase ends, as all nodes in C_2 have less than 3 non-childs. The current graph is shown in Figure 4.8.

In the C_3 phase at depth 0 no independence is found. At depth 1 C-PC finds
$X_3 \perp\!\!\!\perp X_4 \mid X_2$, $X_3 \perp\!\!\!\perp X_5 \mid X_2$, $X_3 \perp\!\!\!\perp X_5 \mid X_4$ and $X_2 \perp\!\!\!\perp X_5 \mid X_4$ and $S_{3,4} = \{X_2\}, S_{3,5} = \{X_2, X_4\}$ and $S_{2,5} = \{X_4\}$. V-structure and orientation rules then allow to orient $X_2 \to X_3$ and $X_4 \to X_5$, as $X_2 \in S_{1,3}$ and $X_4 \in S_{2,5}$. The output of C-PC is shown in Figure 4.9.

Both PC and C-PC recover the correct skeleton. But as can be seen in the accompanying notebook, C-PC has much better arrow performance: 1.0 arrow precision and 0.8 arrow recall, while PC has '*nan*' precision (due to no true and false positives and hence the denominator being zero) and 0.0 recall, as it could not orient any edge. C-PC only fails to orient edge $X_0 - X_1$. In addition, C-PC was able to leverage the C-DAG structure to reduce the number of CI tests. PC needed 153 CI tests whereas C-PC needed only 38.

Example 3: 7-node graph with 3 clusters as v-structure

In the last example some CI tests give a false result. This showcases how PC and C-PC can differ in this case. The ground truth DAG and C-DAG are shown in Figure 4.10, with clusters $C_1 = \{X_0, X_1\}, C_2 = \{X_2, X_3\}, C_3 = \{X_4, X_5, X_6\}$ and C-DAG

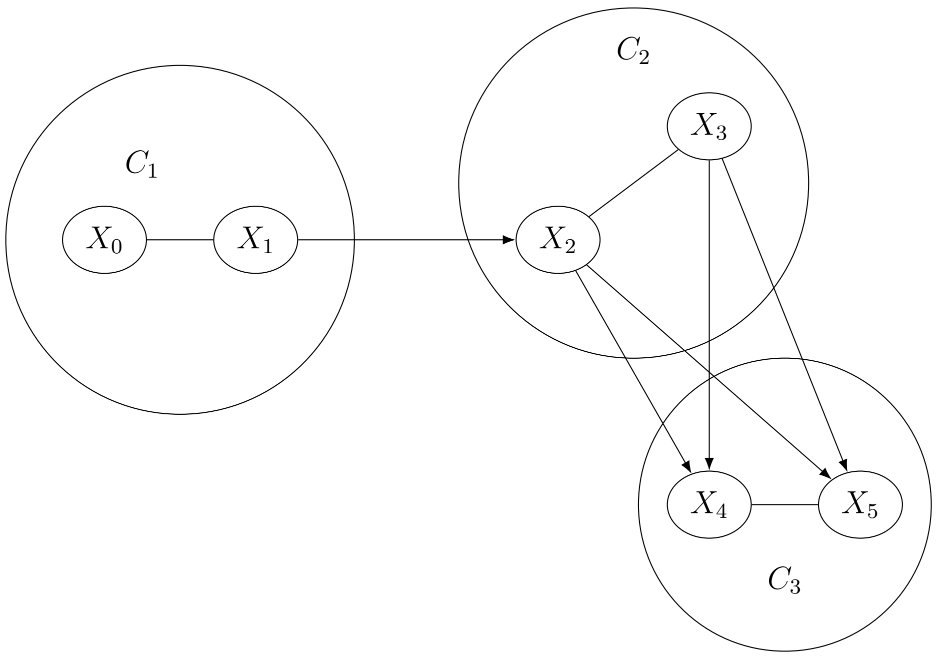

Figure 4.8.: Graph for C-PC after C_1 and C_2 phase in six node example.

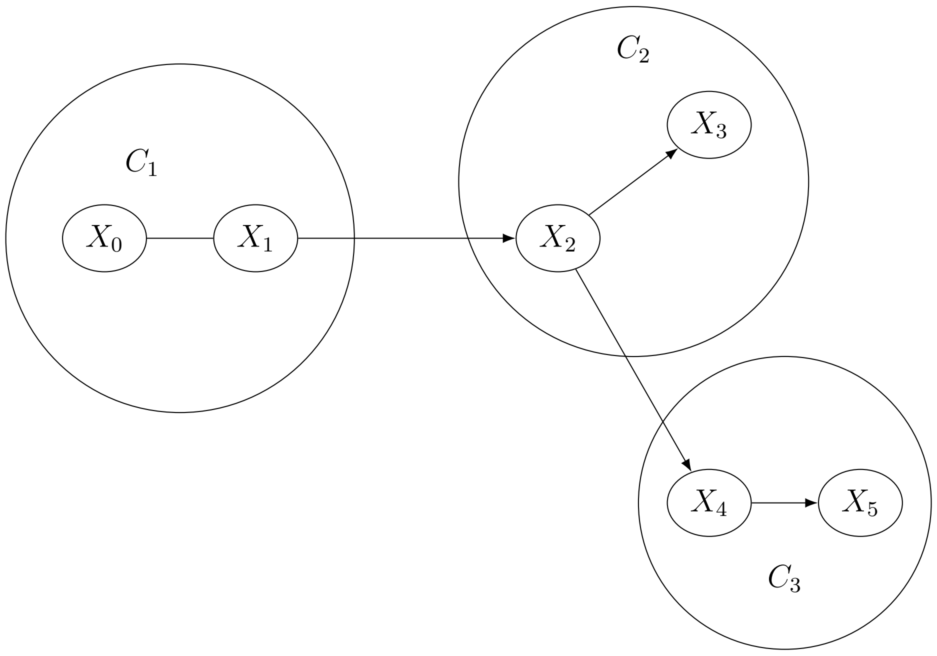

Figure 4.9.: Output of C-PC in six node example.

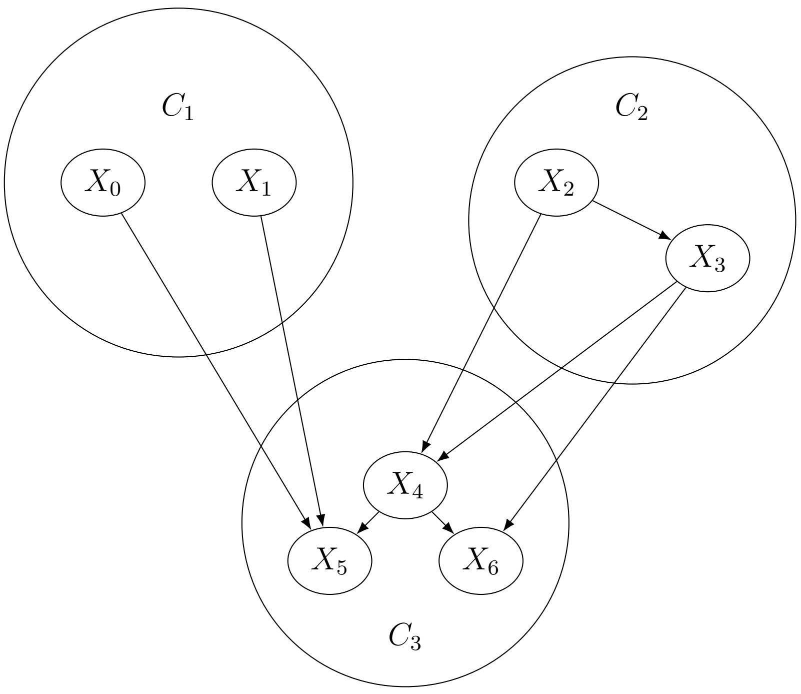

Figure 4.10.: Seven node DAG example.

$C_1 \rightarrow C_2 \leftarrow C_3$.

I will not go into the steps of C-PC in detail here. Importantly, the data was chosen s.t. the following CI tests give the wrong result: $X_2 \perp\!\!\!\perp X_3 \mid X_4$ with a p-value of 0.48, $X_2 \perp\!\!\!\perp X_4 \mid X_6$ with a p-value of 0.61 and $X_3 \perp\!\!\!\perp X_4 \mid X_6$ with a p-value of 0.2. The alpha for this run is 0.1, so for these tests $p > \alpha$ and the edges between X_2, X_3 and X_3, X_4 get erroneously deleted in the PC algorithm. Its output is shown in Figure 4.11.

C-PC in comparison does not perform the CI tests $X_2 \perp\!\!\!\perp X_3 \mid X_4$ and $X_2 \perp\!\!\!\perp X_3 \mid X_6$, because X_4, X_6 are not in the local graph $L_2 = C_2 \cup \bigcup_{C_s \in pa^{G_C}_{C_2}} C_s = C_2 = \{X_2, X_3\}$. Intuitively the C-DAG information says that as X_2, X_3 have no parents, the only way for them to be non-adjacent is if they are unconditionally independent. That is not the case, so no further CI tests are needed, because performing CI tests that put children into the separating set can only open collider paths and not block any active paths.

C-PC is therefore able to avoid erroneously deleting the edge $X_2 - X_3$. It still erroneously deletes edge $X_3 \rightarrow X_4$ though. Together with extra edge orientations, its output is more correct and more informative than PC and is shown in Figure 4.12. In this case PC took 126 CI tests and C-PC took 82.

These examples illustrate some differences between PC and C-PC, but it is difficult to ascertain how C-PC compares to PC in general from those alone. They help in developing an understanding for some of their differences and make one appreciate how C-PC is able to orient extra edges and use less independence tests. But one could also

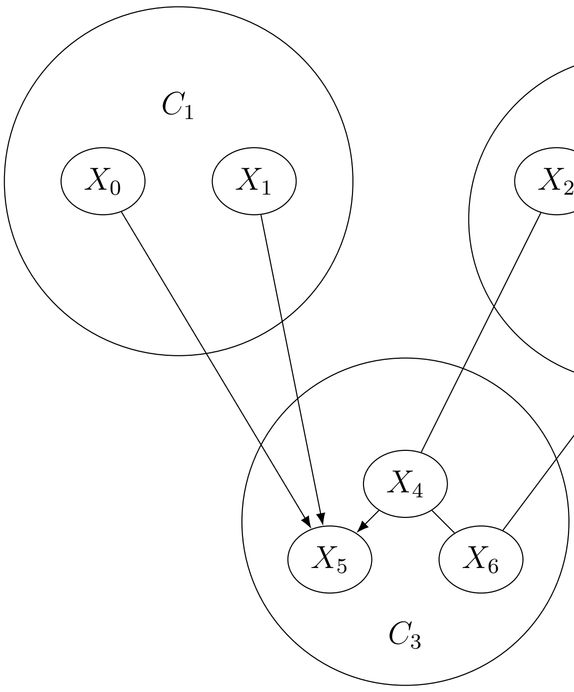

Figure 4.11.: PC output on seven node DAG with erroneous CI tests.

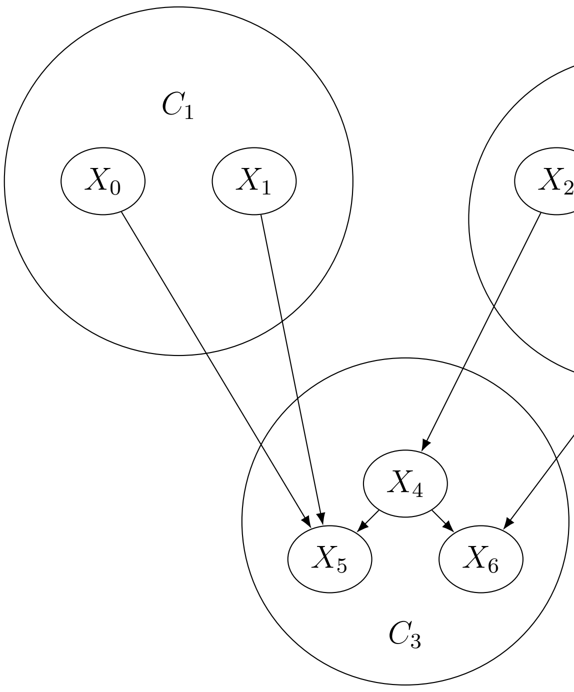

Figure 4.12.: C-PC output on seven node DAG with erroneous CI tests.

construct examples where C-PC performs worse than PC, for example when two false CI tests would cancel each other out. For example, the first CI test falsely does not delete an edge and the second CI test, also false, deletes that edge by accident and therefore cancels out the first mistake, but C-PC does not perform that second CI test. To explore the structural differences between PC and C-PC, larger simulation studies are needed, which will be presented in the following Section 4.3.2.

4.3.2. Simulation studies

This chapter presents the key findings of the two simulation studies that were performed to showcase C-PC and compare it to PC. Further information on how to run a simulation study and evaluate it can be found in the accompanying notebooks "Cluster_PC_simulation_1.ipynb" and "Cluster_PC_simulation_2.ipynb" in the clustercausal package. The last part of "Cluster_PC_examples.ipynb" shows how to manually use the simulation and evaluation functionality. Graphs from the simulation can also be reloaded and looked at, shown at the beginning of "Cluster_PC_simulation_1.ipynb" and "Cluster_PC_simulation_2.ipynb"

The metrics used for evaluation in this chapter are the precision, recall and F1-score as well as the structural Hamming distance (SHD) and number of CI tests performed.

Definition 4.10 (Precision, recall, F1-score). *The precision is defined as*

$$\textit{precision} = \frac{TP}{TP + FP}, \tag{4.2}$$

where TP = true positives and FP = false positives. Positive means the corresponding edge is present and negative means it is absent.
The recall is defined as

$$\textit{recall} = \frac{TP}{TP + FN}, \tag{4.3}$$

where FN = false negative, i.e. an edge was erroneously deleted.
The F1-score is the harmonic mean of recall and precision and encourages balance between the two, as it is zero whenever one of them is zero,

$$\textit{F1-score} = \frac{\textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}}. \tag{4.4}$$

Definition 4.11 (Structural Hamming distance). *The structural Hamming distance (SHD) between graphs G, G' is the number of edge deletions, additions or flips needed to transform G into G'.*

The clustercausal package also allows computation of the structural intervention distance (SID, [PB15])) which is better suited to estimate differences between interventional distributions. It is quite expensive to compute and since smaller experiments showed similar trends between SHD and SID, it is left out of the analysis.

Table 4.1.: Most important parameters for simulation 1. A breakdown of all tunable parameters can be found in Table A.1.

Hyperparameter	Values
Total number of graphs	1750
runs per configuration	10
DAG generation method	Erdos-Renyi
Distribution	gaussian
Alpha for CI test	[0.01, 0.05, 0.1, 0.25, 0.5]
CI test	Fisher-z [Fis+21]
Number of nodes	15
Number of edges	[15, 30, 50, 80, 150]
Number of clusters	[1, 2, 3, 4, 5, 6, 7]
Sample size	1000
Weight range	(-1,2)
Cluster method	'dag' (cf. Appendix A.2.1)

Precision, recall and F1-score are considered w.r.t. adjacency. Since C-PC has access to many more edge orientations, a comparison between arrow precision, recall and F1-Score would have been unfairly lopsided.

The raw data of the simulations can be downloaded under `https://drive.google.com/drive/folders/1EViPZTdyvURlOvlQWfsmtFgypXUiOzym?usp=sharing`.

The first simulation study generated 1750 DAGs total with the Erdos-Renyi graph generation method ([ER59], [Gil59]) from gcastle [Zha+21b]. The probability distribution is gaussian with a linear SEM. All settings of simulation 1 are summarized in Table 4.1.

The second simulation study generated 1080 graphs and can be found in "Cluster_PC_simulation_2.ipynb". It ran the settings shown in Table 4.2. Simulation 2 focused on different DAG generation methods (additionally hierarchical and scale free, see gcastle [Zha+21b]) and different distributions (additionally exponential and gumbel, see gcastle [Zha+21b]).

The key findings are are summarized below. On average in simulation 1

- C-PC has slightly worse adjacency precision than PC (85.1% vs. 87.9%, see Figure 4.13),
- C-PC has better adjacency recall than PC (55.5% vs. 46.4%, see Figure 4.13),
- C-PC has slightly better adjacency F1-score than PC (62.7% vs. 55.3%, see Figure 4.13),
- C-PC has lower SHD than PC (38.2 vs. 44.8, see Figure 4.13),
- C-PC needs less CI tests than PC (3063 vs. 4981, a reduction by 38.5%, see Figure

Table 4.2.: Most important parameters for simulation 2. A breakdown of all tunable parameters can be found in Table A.1.

Hyperparameter	Values
Total number of graphs	1080
runs per configuration	1
DAG generation method	Erdos-Renyi, hierachical and scale free
Distribution	exponential, gaussian and gumbel
Alpha for CI test	[0.01, 0.05, 0.1, 0.25, 0.5]
CI test	Fisher-z [Fis+21]
Number of nodes	15
Number of edges	[15, 30, 50, 80]
Number of clusters	[1, 2, 3, 4, 5, 6]
Sample size	1000
Weight range	(-1,2)
Cluster method	'dag' (cf. Appendix A.2.1)

4.13),

- all of these metrics improve with increasing numbers of clusters (see Figures 4.14 to 4.18).
- C-PC is strictly better than PC in terms of recall for different alphas as can be seen in Figures 4.19 and 4.20, so C-PC vs. PC is not comparable to a precision-recall trade-off from changing alpha.
- Simulation 2 showed no significant difference of the aforementioned trends across different DAG generation methods and distributions (see Figure 4.21).

Table 4.3.: Base PC metrics

Metric	Value
Adj. precision	**87.9%**
Adj. recall	46.5%
Adj. F1-score	55.3%
SHD	44.8
CI tests	4981

Table 4.4.: Cluster-PC metrics

Metric	Value
Adj. precision	85.1%
Adj. recall	**55.5%**
Adj. F1-score	**62.7%**
SHD	**38.2**
CI tests	**3063**

Figure 4.13.: Comparison of Base PC and Cluster-PC metrics for simulation 1. Cluster-PC is slightly worse in precision, i.e. it keeps slightly more wrong edges. Cluster-PC is superior in precision (less edges falsely removed) and SHD and reduces the number of needed CI tests.

Table 4.5.: Base PC and Cluster-PC metrics for different no. of clusters

No. of clusters	Adj. precision	Adj. recall	Adj. F1-score	SHD
1 = base PC	**87.4%**	46.8%	55.3%	45.1
2	85.8%	50.1%	58.7%	42.0
3	83.9%	54.5%	61.3%	39.7
4	84.1%	56.4%	63.1%	37.6
5	84.3%	58.3%	65.1%	36.1
6	84.4%	59.7%	66.4%	34.5
7	86.1%	**62.3%**	**68.8%**	**32.5**

Figure 4.14.: Simulation 1: The adjacency precision is slightly higher for one cluster, i.e. Base PC. Recall, F1-score and SHD improve significantly with increasing number of clusters. The metrics for Base PC for 2,...,7 clusters are not shown explicitly, they are very similar to the one cluster case.

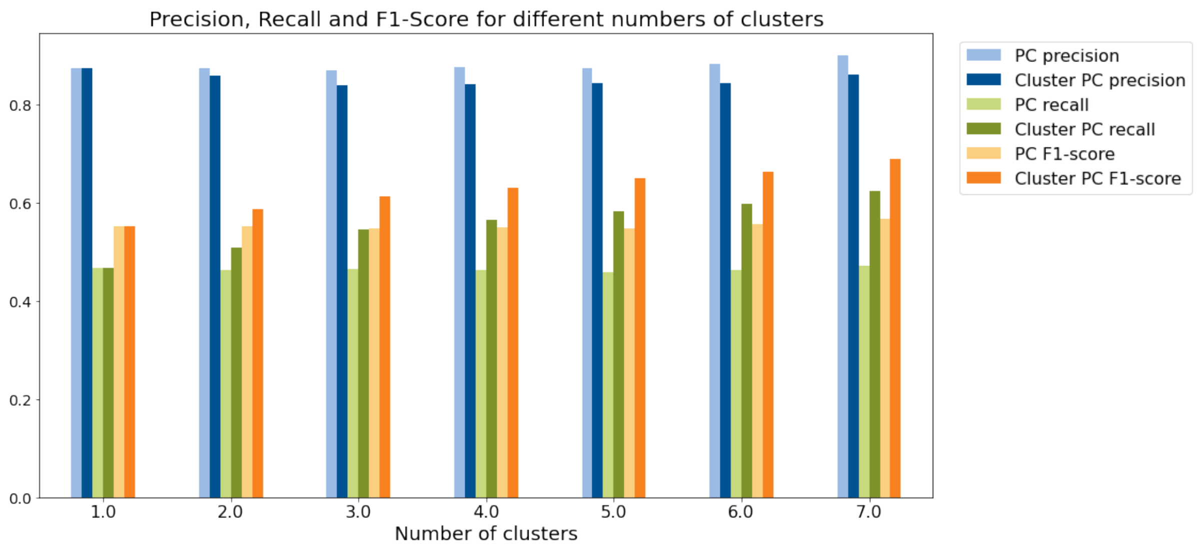

Figure 4.15.: Simulation 1: Precision, recall and F1-score for different numbers of clusters.

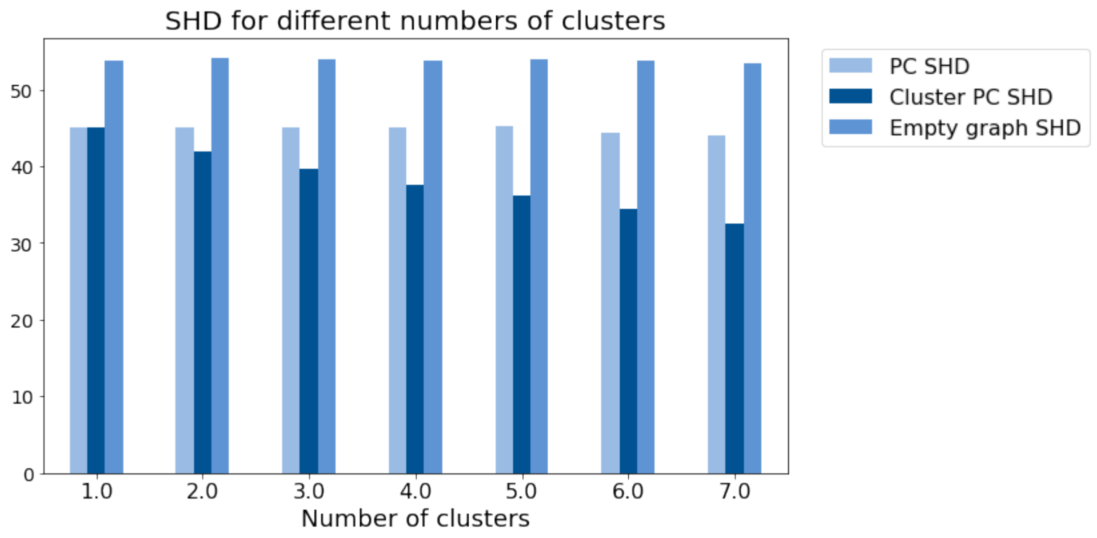

Figure 4.16.: Simulation 1: SHD for different numbers of clusters.

Table 4.6.: Base PC and Cluster-PC metrics for different no. of clusters

No. of clusters	Base PC CI tests	Cluster-PC CI tests
1	4985.0	4985.0
2	5022.0	3855.0
3	4974.0	3140.0
4	5057.0	2781.0
5	4993.0	2436.0
6	4923.0	2201.0
7	4914.0	2039.0

Figure 4.17.: Simulation 1: The needed number of CI test goes down significantly with increasing number of clusters.

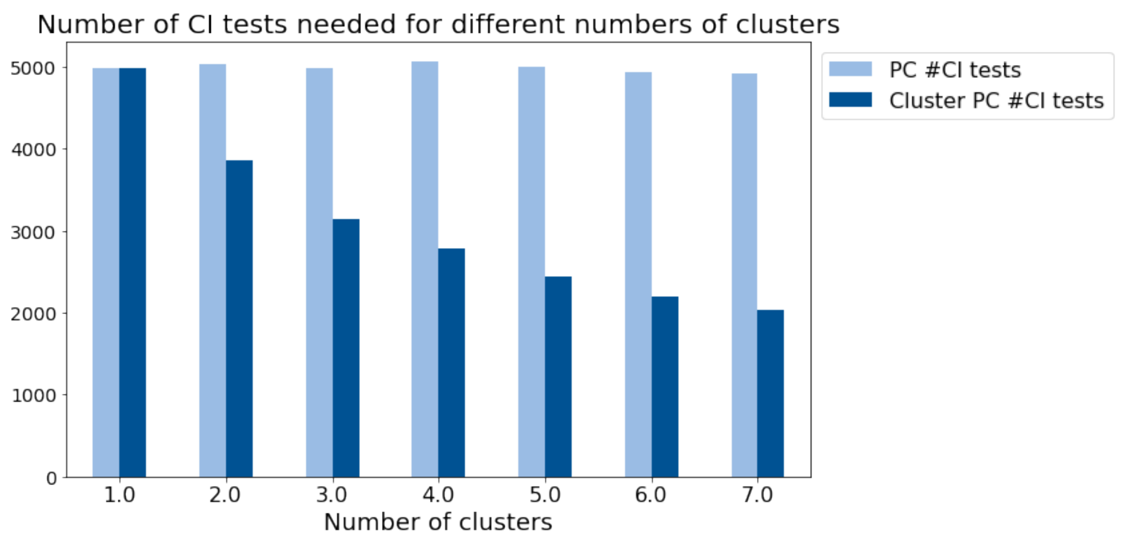

Figure 4.18.: Simulation 1: CI tests for different numbers of clusters.

Table 4.7.: Base PC metrics for different alpha values

Alpha	Adj. precision	Adj. recall	Adj. F1-score	SHD
0.01	**92.8%**	42.0%	53.1%	44.5
0.05	91.4%	44.1%	54.7%	44.2
0.10	89.8%	46.6%	56.4%	43.9
0.25	85.6%	48.1%	56.0%	45.3
0.50	80.3%	51.4%	56.4%	46.1

Table 4.8.: Cluster-PC metrics for different alpha values

Alpha	Adj. precision	Adj. recall	Adj. F1-score	SHD
0.01	91.4%	51.8%	63.0%	37.5
0.05	89.1%	54.7%	64.5%	36.6
0.10	87.2%	57.3%	**65.7%**	**35.7**
0.25	81.4%	58.7%	63.8%	37.5
0.50	74.6%	**62.5%**	62.6%	38.1

Figure 4.19.: Comparison of Base PC and Cluster-PC metrics in simulation 1 for different alpha values. Cluster-PC shows consistently better recall and therefore F1-score across all alpha values, this is visualized in Figure 4.20.

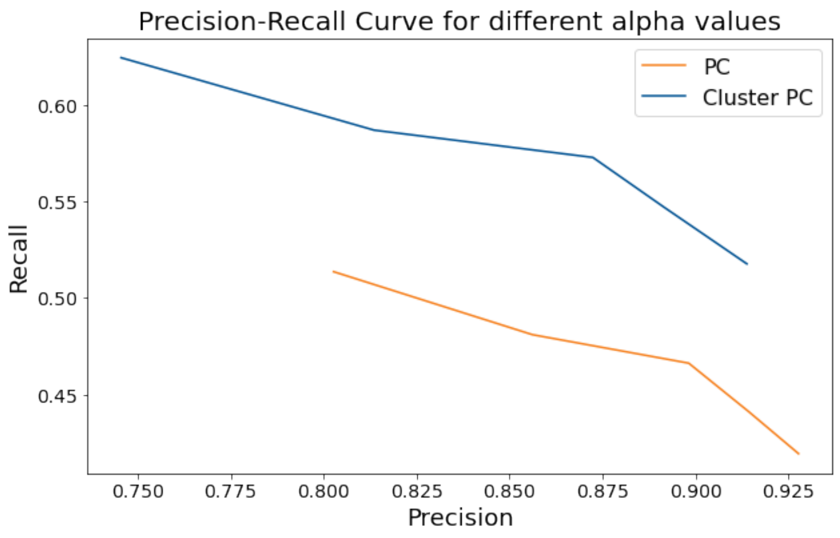

Figure 4.20.: Precision, recall and F1-score for different alphas. Cluster-PC dominates base PC w.r.t. recall.

The performance of Cluster-PC improves with more clusters, as one would expect. Surprisingly, Cluster-PC has worse precision than base PC, i.e. it does not delete as many edges as it should. This is probably due to it performing less CI tests, so base PC deletes some edges due to CI tests that Cluster-PC never performs, compare also to the last example of Section 4.3.1. On the other hand, its special structure allows Cluster-PC to have much better recall, as it is less likely to delete an edge erroneously, as it performs less potentially faulty CI tests. This way, Cluster-PC is a more cautious version of base PC, it deletes less edges, may therefore leave some false positives, but is much less likely to have false negatives. For causal inference, this may be desirable as it is less bad to have an edge too much than an edge too few, the latter would cause one to conclude no influence, while the former cautions one about a potential influence. Figure 4.20 also shows that this improvement cannot be achieved in base PC by altering the edge-deletion hyperparameter alpha. Cluster-PC is strictly better in recall for all alpha values that were considered.

In conclusion Cluster-PC has a better F1-score and SHD. The two simulation studies show that the results presented here are very robust across various different hyperparameter settings. Together with Cluster-PC using less CI tests, these simulation studies suggest that Cluster-PC is a desirable improvement if the background knowledge allows for construction of a C-DAG.

Table 4.9.: Base PC metrics for different distributions and DAG methods

Distribution	Adj. precision	Adj. recall	Adj. F1-score	SHD
Exponential	85.8%	46.5%	55.7%	40.3
Gaussian	87.0%	46.5%	56.2%	40.1
Gumbel	86.6%	46.2%	55.8%	40.4
DAG method	**Adj. precision**	**Adj. recall**	**Adj. F1-score**	**SHD**
Erdos-Renyi	84.9%	54.5%	61.8%	32.1
Hierarhical	90.3%	24.8%	38.7%	65.6
Scale free	84.2%	60.0%	67.3%	23.1

Table 4.10.: Cluster-PC metrics for different distributions and DAG methods

Distribution	Adj. precision	Adj. recall	Adj. F1-score	SHD
Exponential	82.8%	56.4%	63.5%	34.3
Gaussian	83.7%	56.4%	63.8%	33.9
Gumbel	83.1%	55.7%	63.1%	34.6
DAG method	**Adj. precision**	**Adj. recall**	**Adj. F1-score**	**SHD**
Erdos-Renyi	80.9%	63.5%	67.6%	27.4
Hierarhical	89.3%	36.7%	51.6%	56.0
Scale free	79.6%	68.4%	71.3%	19.5

Figure 4.21.: Simulation 2: Comparison of Base PC and Cluster-PC metrics for different distributions and DAG generation methods. The trends seen before of slightly worse precision for Cluster-PC and better recall, F1-score and SHD compared to base PC are also visible here.

5. C-ADMGs for the FCI algorithm

As C-ADMGs can contain bidirected edges, a natural question is if it is possible to extend the FCI algorithm from Section 2.3.3 to C-ADMGs in a similar way as was possible for the PC algorithm for C-DAGs. This chapter explores this idea of the Cluster-FCI algorithm (C-FCI). To keep this chapter tractable, I assume no selection variables, their inclusion could be a topic for future research.

5.1. C-ADMG to initial graph

For C-PC it was possible to use an MPDAG (a graph pruned and oriented according to the C-DAG) as initial graph for a modified PC algorithm. For C-ADMGs this is also possible, albeit needing an extra step due to 1) the potential presence of inducing paths and 2) C-ADMGs and ADMGs not being the correct graph for FCI to operate on.

Definition 5.1 (Partial mixed graph for C-ADMG). *The partial mixed graph $G_{pm} = (V, E)$ of a C-ADMG G_C is a graph with possibly three kind of edge marks (and five kinds of edges: $\rightarrow$, $\leftrightarrow$, $\circ\!-$, $\circ\!\!-\!\!\circ$, $\circ\!\!\rightarrow$), such that*

(i) for $X_i, X_j \in C_i$ it is $X_i \circ\!\!-\!\!\circ X_j \in E$,

(ii) for $X_i \in C_i, X_j \in C_j$ with $C_i \rightarrow C_j$ and $C_i \not\leftrightarrow C_j$ it is $X_i \rightarrow X_j \in E$,

(iii) for $X_i \in C_i, X_j \in C_j$ with $C_i \rightarrow C_j$ and $C_i \leftrightarrow C_j$ it is $X_i \circ\!\!\rightarrow X_j \in E$,

(iv) for $X_i \in C_i, X_j \in C_j$ with C_i, C_j not adjacent and connected by an inducing path, if

- *$C_i \in anC_j^{G_C}$ it is $X_i \rightarrow X_j \in E$,*
- *$C_j \in anC_i^{G_C}$ it is $X_i \leftarrow X_j \in E$,*
- *$C_i \notin anC_j^{G_C}$ $C_j \notin anC_i^{G_C}$ and it is $X_i \leftrightarrow X_j \in E$.*

The partial mixed graph G_{pm} for a C-ADMG G_C will be the starting graph for the Cluster-FCI algorithm (cf. Algorithm 11) and is produced from G_C by using Algorithm 10.

Definition 5.2 (Compatibility of MAGs and partial mixed graphs with C-ADMGs). *A partial mixed graph G_{pm} is called compatible with C-ADMG G_C if none of its edges contradict the edges induced by G_C together with Algorithm 10. A MAG G_M is* *called compatible with C-ADMG G_C if none of its edges contradict the edges induced by G_C together with Algorithm 10.*

Algorithm 10 C-ADMG to partial mixed graph transformation for C-FCI

```
Require: C-ADMG $G_C$ with $C = \{C_1, ...C_m\}$
 1: form undirected graph $G$ over $V = \bigcup_{i=1}^{M} C_i$
 2: for all clusters $C_i$ in $G_C$ do
 3:     add $X_i \circ\!\!-\!\!\circ X_j$ to $G$
 4: for all adjacent clusters $C_i, C_j$ in $G_C$ do
 5:     for $X_i \in C_i, X_j \in C_j$ do
 6:         if $C_i \rightarrow C_j$ and $C_i \not\leftrightarrow C_j$ then
 7:             add $X_i \rightarrow X_j$ to $G$
 8:         if $C_i \leftarrow C_j$ and $C_i \not\leftrightarrow C_j$ then
 9:             add $X_i \leftarrow X_j$ to $G$
10:         if $C_i \rightarrow C_j$ and $C_i \leftrightarrow C_j$ then
11:             add $X_i \circ\!\!\rightarrow X_j$ to $G$
12:         if $C_i \leftarrow C_j$ and $C_i \leftrightarrow C_j$ then
13:             add $X_i \leftarrow\!\!\circ X_j$ to $G$
14:         if $C_i \not\rightarrow C_j$, $C_i \not\leftarrow C_j$ and $C_i \leftrightarrow C_j$ then
15:             add $X_i \leftrightarrow X_j$ to $G$
16: for all non-adjacent clusters $C_i, C_j$ connected by an inducing path in $G_C$ do
17:     if $C_i \in an^{G_C}_{C_j}$ then
18:         for $X_i \in C_i, X_j \in C_j$ do
19:             add $X_i \rightarrow X_j$ to $G$
20:     if $C_j \in an^{G_C}_{C_i}$ then
21:         for $X_i \in C_i, X_j \in C_j$ do
22:             add $X_i \leftarrow X_j$ to $G$
23:     if $C_i \notin an^{G_C}_{C_j}$ and $C_j \notin an^{G_C}_{C_i}$ then
24:         for $X_i \in C_i, X_j \in C_j$ do
25:             add $X_i \leftrightarrow X_j$ to $G$
26: return graph partial mixed graph $G_{pm} = G$
```

Lines 2-15 of the procedure add all edges to G that can be directly read off from G_C. When there is a directed and a bidirected edge between clusters, the edge $\circ\!\!\rightarrow$ is added. Lines 16-25 are similar to Algorithm 2 which transforms a DAG into a MAG. Here it is used to add edges between nodes that are not adjacent in G but are potentially not m-separable, due to the potential presence of an inducing path .

Note that the resulting graph must not necessarily be ancestral, as the C-ADMG could contain almost directed cycles. Hence the output of Algorithm 10 can also contain almost directed cycles. [HE20] give an algorithm to transform an ADMG into a MAG, which however is not desirable in this situation. Consider the graph in Figure 5.1. The MAG G_M over $\{X_1, ...X_5\}$ is ancestral, but the corresponding C-ADMG G_C with

$C = \{C_1, C_2, C_3\}, C_1 = \{X_1, X_2\}, C_2 = \{X_3\}, C_3 = \{X_4, X_5\}$ is not, due to the almost directed cycle $C_3 \leftrightarrow C_1 \to C_2 \to C_3$. Even though there is an almost directed cycle in G_C, G_M is compatible with G_C.

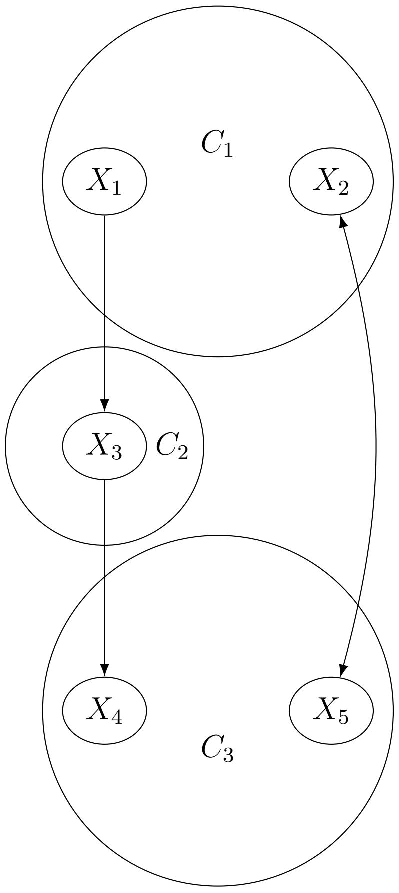

Figure 5.1.: Example of non-ancestral C-ADMG.

It would be a mistake to transform the C-ADMG into a MAG: that would remove the edge $C_1 \leftrightarrow C_2$ and change it to $C_1 \to C_2$. The edge between X_2, X_5 would inherit this orientation and be $X_2 \to X_5$, in contradiction with the original MAG G_M. An additional step in C-FCI will be needed to ensure that it outputs a PAG, i.e. at the end of C-FCI, any almost directed cycles, the bidirected edge is replaced by a directed edge.

It is possible to change the C-FCI algorithm to not output a PAG (Non-PAG C-FCI), by not re-orienting almost directed cycles. This can be an improvement, as it increases the information contained in its output. To see why, imagine that the graph in Figure 5.1 also contains the edge $X_1 \leftrightarrow X_4$. FCI would at best be able to return the edge $X_1 \to X_4$, due to the almost directed cycle $X_4 \leftrightarrow X_1 \to X_3 \to X_4$. Non-PAG C-FCI in contrast will be able to return $X_1 \leftrightarrow X_4$, a more informative result. As this leaves the well explored field of using ancestral graphs for causal discovery, I decided to not pursue this further. Whether the orientation rules of [Zha08b] are complete for Non-PAG C-FCI, and whether it is sound, is a topic for future research.

5.2. C-FCI algorithm

It is now possible to state the C-FCI algorithm, which uses strategies similar to C-PC. A small adjustment in the definition of parents, children and non-childs is needed for graphs with circle edge marks. For an edge $X \circ\!\!\rightarrow Y$ it makes sense to regard X as a parent of Y, as the edge $X \circ\!\!\rightarrow Y$ was introduced in the partially mixed graph because the clusters $X \in C_x, Y \in C_y$ have the edges $C_x \rightarrow C_y$ connecting them.

Definition 5.3 (**Updated pa, ch and nch**). *In the following an edge $X \circ\!\!\rightarrow Y$ encodes $X \in pa_Y$ and $Y \in ch_X$. $X \in nch_Y$ means either $X \circ\!\!-\!\!\circ Y$, $X \circ\!\!\rightarrow Y$ or $X \leftrightarrow Y$. The same is defined for an and de.*

Algorithm 11 Cluster-FCI algorithm

```
Require: Joint distribution $P_O$ of $d$ observed variables, independence oracle, C-ADMG
         $G_C = (V_C, E_C), C = \{C_1, ..., C_r\}$ with clusters in topological ordering (w.r.t.
         directed edges) compatible with ground truth MAG
 1: Construct graph $G$ from $G_C$ with Algorithm 10
 2: for $m \in [r]$ set $L_m := C_m \cup \bigcup_{C_s \in pa^{G_C}_{C_m}} C_s \cup \bigcup_{C_s \in sib^{G_C}_{C_m}} C_s$ do    ▷ cluster phase for $C_m$
 3:     for $k = 0, ..., |L_m| - 2$ do                                  ▷ with local graph $L_m$
 4:         $del_E = \emptyset$                                          ▷ set for edge deletions after depth phase
 5:         for each adjacent $X_j \in C_m$, $X_i \in pa_{X_j}$ do
 6:             for all $S \subset nch_{X_j} \setminus \{X_i\}$ and $|S| = k$ do
 7:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
 8:                     $del_E \leftarrow (X_i \circ\!\!\rightarrow X_j)$ or $del_E \leftarrow (X_i \rightarrow X_j)$
 9:                     $S_{i,j} \leftarrow S$
10:         for each adjacent $X_j \in C_m$, $X_i \in sib_{X_j}$ do
11:             for all $S \subset nch_{X_j} \setminus \{X_i\}$ or $S \subset nch_{X_i} \setminus \{X_j\}$ and $|S| = k$ do
12:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
13:                     $del_E \leftarrow (X_i \leftrightarrow X_j)$
14:                     $S_{i,j} \leftarrow S$
15:         for each adjacent $X_i, X_j \in C_m$ do
16:             for all $S \subset nch_{X_j} \setminus \{X_i\}$ or $S \subset nch_{X_i} \setminus \{X_j\}$ and $|S| = k$ do
17:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
18:                     $del_E \leftarrow (X_i \circ\!\!-\!\!\circ X_j)$
19:                     $S_{i,j} \leftarrow S$
20:         delete edges from $del_E$ in $G$
```

```
21: for all unshielded triples $(X_i, X_j, X_k)$ do
22:     if $X_j \notin S_{i,k}$ then
23:         if $X_i \ast\!\!-\!\!\circ X_j \circ\!\!-\!\!\ast X_k$ then
24:             orient $X_i \ast\!\!-\!\!\circ X_j \circ\!\!-\!\!\ast X_k$ as $X_i \ast\!\!\rightarrow X_j \leftarrow\!\!\ast X_k$ in $G$
25:         if $X_i \ast\!\!\rightarrow X_j \circ\!\!\rightarrow X_k$ then
26:             orient $X_i \ast\!\!\rightarrow X_j \circ\!\!\rightarrow X_k$ as $X_i \ast\!\!\rightarrow X_j \leftrightarrow X_k$ in $G$
27: for all nodes $X_i$ in $G$ do
28:     compute $pds(G, X_i, \cdot)$ as defined in Definition 2.39
29:     for all nodes $X_j \in adj^G_{X_i}$ do
30:         for $k = 0, ..., d-2$ do
31:             for $|S| \subset pds(G, X_i, \cdot) \setminus \{X_j\}$ with $|S| = k$ do
32:                 if $X_i \perp\!\!\!\perp X_j \mid S$ then
33:                     delete edge $X_i \ast\!\!-\!\!\ast X_j$ from $G$
34:                     let $S_{i,j} = S_{j,i} = S$
35: reorient all edges according to C-ADMG $G_C$ (as in Algorithm 10 but only orienting
    edges, not adding edges)
36: for any almost directed cycle $X_l \leftrightarrow X_1 \rightarrow ... \rightarrow X_{l-1}$ orient $X_l \leftrightarrow X_1$ to $X_l \rightarrow X_1$
37: use rules R0-R4, R8-R10 of [Zha08b] to orient as many edge marks as possible
38: return PAG $G = (V, E)$
```

Instead of starting with a fully connected $\circ\!\!-\!\!\circ$ graph, the first part of C-FCI constructs a partial mixed graph with edges according to the given C-ADMG G_C via Algorithm 10. Then it proceeds with the skeleton discovery via adjacency separating sets, as in the standard FCI. Here it can use some of the C-ADMG structure and work down the topological ordering (implied by the directed edges) of the clusters, like the C-PC algorithm. This works because nodes in lower clusters are on paths that contain colliders, which are blocked by not conditioning on them.

Each cluster phase for a cluster C_m works similar to C-PC. First connections to parent clusters and sibling clusters get considered, and then connections within clusters. Afterwards, the normal FCI continues with orienting triples and then searches for more edge deletions with possibly d-separating sets *pds*. These *pds* sets can be computed as in FCI, in FCI they are calculated based on the adjacency and path structure of the current graph. The same can be done in C-FCI. Instead of orienting all edges as $\circ\!\!-\!\!\circ$ before the orientation rules, C-FCI can use the orientations induced by the C-ADMG. Almost directed cycles are re-oriented to make the output a PAG. Then it uses the orientation rules from [Zha08b], with the exception of rules R5-R7, as these deal with selection variables which were assumed to be absent.

For C-FCI, a similar situation to Theorem 4.8 arises:

Theorem 5.4 (**C-FCI: no problem with sepsets**). *The C-FCI algorithm in Algorithm 11 not adding separating sets for removed edges does not lead to wrong v-structure orientations.*

Proof. Apart from what was already proven in Theorem 4.8, there are two potential triples that could lead to problems. These are $X_i {*\!\!\rightarrow} X_j {\circ\!\!\rightarrow} X_k$ and $X_i \leftarrow\!\!\circ X_j {\circ\!\!\rightarrow} X_k$ with X_i, X_j, X_k all in different clusters. In both cases, the clusters of $X_i \in C_i$ and $X_k \in C_k$ will be connected to each other due to a potentially inducing path. In the first case, $C_i \rightarrow C_j \rightarrow C_k$ and $C_j \leftrightarrow C_k$. This is an inducing path with $C_i \in an^{G_C}_{C_k}$, so $C_i \rightarrow C_k$ will be included in G_{pm} by Algorithm 10. In the second case, $C_i \leftarrow C_j \rightarrow C_k$ and $C_i \leftrightarrow C_j, C_j \leftrightarrow C_k$. That is also an inducing path with $C_i \notin an^{G_C}_{C_k}$ and $C_k \notin an^{G_C}_{C_i}$, so $C_i \leftrightarrow C_k$ will be included in G_{pm} by Algorithm 10. In both cases, X_i and X_k were adjacent at some point during C-FCI, so CI tests of the form $X_i \perp\!\!\!\perp X_k \mid S$ with $X_j \in S$ were performed and it is decidable wheter $X_j \in S_{i,k}$. Hence all v-structure will be oriented correctly. □

One can see that the C-FCI algorithm is sound: any edge that the FCI algorithm would remove is either removed by the C-ADMG or removed by a CI test. Furthermore C-FCI cannot orient an edge wrongly, either it does the same as FCI or orients an edge according to C-ADMG or orientation rules resulting from the C-ADMG orientations. Either way, the edge cannot be oriented wrongly.

Theorem 5.5 (Soundness of C-FCI). *If the C-ADMG G_C is compatible with ground truth MAG G_M, C-FCI is sound in the sense that an edge between any nodes X_i, X_j is present in MAG G_m if and only if it is present in the PAG G_P output by C-FCI. Any arrow or tail edge mark in G_P is also present in G_M.*

Proof. One can see the first phase of C-FCI that builds the mixed graph based on Algorithm 10 as an additional 'oracle' phase. Instead of asking an independence oracle whether an edge needs to be removed, the C-ADMG oracle is queried. And since C-FCI checks all remaining edges with a superset of an m-separating set (sets in *nch* and *dsep*), it recovers only and exactly only the edges in the ground truth MAG.
Any arrow or tail orientation in C-FCI either comes from FCI and its orientation rules, assumed to be correct C-ADMG orientations or a combination of those two. In all cases, an arrow or tail edgemark is oriented correctly in G_P, i.e. it is also present in G_M. □

Whether C-FCI is complete in the sense that its output is maximally informative, i.e. all circles respond to variant edge marks in the restricted Markov equivalence class of $[M(G_M), bk(G_C)]$, is conjectured to be true. A formal proof however is a bigger endeavour and was not attempted in this thesis. C-FCI is closed under the orientation rules R0-R4 and R8-R10 from [Zha08b], but it may be possible that additional orientation rules using C-ADMG knowledge exist and that they are needed to recover the restricted Markov equivalence class $[M(G_M), bk(G_C)]$ fully.

This chapter presented the C-FCI algorithm which uses C-ADMGs to achieve a more informative version of the FCI algorithm. Similar to C-PC it makes use of the topological structure of the C-ADMG to reduce number of necessary CI tests and potentially avoids erroneous CI tests. The C-FCI algorithm was implemented in the clustercausal package. Experiments on handcrafted graphs can be seen in the notebook

"Cluster_FCI_examples.ipynb". In small experiments, the algorithm performed well, but in contrast to C-PC it was not extensively bug-tested and simulated with, so it is considered in beta stage.

6. Conclusion

In this thesis, I proposed to use C-DAGs as background knowledge to enhance constraint based causal discovery algorithms. To my knowledge, this is the first work to do so.

I presented the relevant literature on graphical statistical models, causality and causal discovery. Then I positioned C-DAGs and C-ADMGs within the existing literature on background knowledge. In particular, I showed that C-DAGs and C-ADMGs can be expressed as a boolean combination of pairwise background knowledge constraints, which allows for comparison across methods and better transferability of theoretical results. C-DAGs and C-ADMGs were shown to possess advantages over other established aggregate background knowledge frameworks, namely tiered background knowledge and typing assumption. As could be seen, C-DAGs and C-ADMGs occupy a sweet spot between the two, allowing for a useful variable ordering compared to the typing assumption, while also being more flexible in the modeling of group relationships compared to tiered background knowledge.

To showcase these advantages practically, I developed the python package clustercausal (`https://github.com/JanMarcoRuizdeVargas/clustercausal`), which implements the Cluster-PC and the Cluster-FCI algorithm. These two are an adaption of the PC and FCI algorithm respectively, using the C-DAG or C-ADMG structure to work faster and more accurately compared to the baseline versions. The package also allows for large scale simulation studies and their evaluation with respect to many common metrics in causal discovery. The simulation studies suggest a significant and robust improvement of Cluster-PC over the baseline PC algorithm.

In this thesis, C-ADMGs where used for constraint based causal discovery. Future research into incorporating other methods could be interesting. Decomposable scores might allow one to parallelize an algorithm like GES [Chi02], running score based search on a per cluster basis and later aggregating the edges. Furthermore, it might be possible to encode C-DAG or C-ADMG assumptions via constraints on the adjacency matrix and hence use continuous optimization methods such as DAGs with no TEARS [Zhe+18]. In C-FCI, selection variables were not considered. The ramifications of their inclusion is an interesting topic too. A proof of completeness of C-FCI would be a welcome insurance that its output is maximally informative. Lastly, whether Non-PAG C-FCI is sound and complete can also be a future research topic.

A. Appendix: clustercausal package

This chapter gives a brief overview on the clustercausal package (url at `https://github.com/JanMarcoRuizdeVargas/clustercausal`) that I developed to accompany this thesis. An installation guide can be found in the README.mf of the package.

A.1. File structure

The file structure of the clustercausal package is as follows:

```
clustercausal
 ├─ algorithms
 │   ├─ ClusterFCI.py
 │   └─ ClusterPC.py
 ├─ clusterdag
 │   └─ ClusterDAG.py
 ├─ experiments
 │   ├─ configs
 │   ├─ _results
 │   ├─ ExperimentRunner.py
 │   ├─ Evaluator.py
 │   ├─ run_experiment.py
 │   ├─ run_gridsearch.py
 │   ├─ Simulator.py
 │   └─ Utils.py
 ├─ tests
 │   ├─ test_cdag.py
 │   ├─ test_cluster_fci.py
 │   ├─ test_cluster_pc.py
 │   ├─ test_evaluator.py
 │   ├─ test_simulator.py
 │   └─ test_utils.py
 └─ utils
     └─ Utils.py
```

The algorithms folder contains the Cluster-PC and Cluster-FCI algorithm (the latter is in an exploratory stage). The clusterdag folder contains the Cluster-DAG class to handle C-DAG construction. The experiments folder contains functionality to run, save and evaluate experiments. The tests folder contains test cases to ensure changes to the codebase do not break existing functionality. The utils folder contains code to construct handcrafted graphs and datasets.

A.2. Simulation and evaluation

A simulation can be run by configuring a config.yaml file in the configs folder, an overview of possible configurations can also be found in 'cluster_pc_all_param_configs.yaml'. Afterwards one should edit the file path in run_gridsearch.py to use the previously edited yaml file and run

```
python clustercausal\experiments\run_gridsearch.py
```

in the console.

The configurable settings are listed in Table A.1. The experiments are saved in the _results folder.

A.2.1. How C-DAGs are generated

The clustercausal\experiments\Simulator.py handles C-DAG generation. The method I used for my simulation studies was 'dag'. In that case the method Simulator.generate_dag() is used to generate a DAG and afterwards the method Simulator.generate_clustering() generates a clustering by slicing up the topological ordering into n_cluster slices of random size.

For example, if the DAG has ten nodes and the number of clusters is three, the Simulator.generate_clustering() will select two numbers between $[1, n\_clusters]$, say four and ten. This means the first cluster will include the first three nodes in the topologiacl ordering, the second cluster contains nodes four to nine and the third cluster contains node ten.

If the 'cdag' method is used, the Simulator.generate_dag_via_clusters() is used, which first generates an Erdos-Renyi graph for the clusters, for example of size three again. Then it generates nodes for each cluster so that they sum up to the desired node number, say ten again. Then the graph is built according to the generated cluster graph and nodes, and some edges from that graph are dropped out, that probability is influenced by the n_edges parameter.

A.2.2. Overview on evaluation and experiment running

The Evaluator.py calculates metrics to be used for evaluation. The Evaluator.get_causallearn_metrics() function calculates the arrow confusion (includes true/false positives, true/false negatives, precision, recall, F1-score), which computes how often arrows were correctly predicted and the adjacency confusion (includes true/-false positives, true/false negatives, precision, recall, F1-score), which computes how often edges were correctly predicted. The Evaluator.get_shd() function function computes the structural Hamming distance (SHD) using the causal-learn package. The Evaluator.get_sid_bounds() function calculates the structural intervention distance (SID) bounds for a CPDAG. The Evaluator.get_cluster_pruned_benchmark() function removes edges from the base PC graph that could be deleted by pruning according to the C-DAG. In experiments however, that showed to have no effect, i.e. PC already removed edges between non-adjacent clusters. The Evaluator.get_cluster_connectivity() function calculates how many of the clusters are connected by an edge and returns a score between 0 and 1. If the score is 1, the C-DAG is fully connected, i.e. all possible edges are present, and if the score is 0, the C-DAG contains no edges.

The ExperimentRunner.py also saves the number of CI tests in addition to all simulation settings and base PC and Cluster-PC metrics. Furthermore, it saves the edge_ratios, i.e. how many of the intra cluster edges, inter cluster edges and inter cluster edges including disconnected clusters are present, expressed as a number from 0 to 1. Intra cluster edges is the number of edges within clusters divided by the number of possible edges within clusters. Inter cluster edges is the number of edges between clusters divided by the number of possible edges between clusters *that are adjacent.* Inter cluster edges with disconnected clusters is the number of edges between clusters divided by the number of possible edge between all clusters, including clusters that are not adjacent. Every experiment also saves the corresponding graphs which can be loaded and looked at using load_experiment_graphs() from Utils.py.

A.3. Conclusion

This chapter presented the clustercausal package and explained its core components. It provides an easy to use class to handle C-DAG construction and allows for manual handcrafting of graphs and data as well as simulations according to specified parameters. The functionality also includes running, saving and evaluation of simulation studies with many different settings and metrics.

Table A.1.: Parameters for the simulation

Hyperparameter	Values
discovery_alg	discovery algorithm, for now only ClusterPC
runs_per_configuration	1 or more, how often one configuration setting is run
n_nodes	number of nodes in the DAG
n_edges	approximate number of edges in the DAG
dag_method	the method used to generate the DAG, can be Erdos-Renyi, hierarchical or scale-free from gcastle [Zha+21b]
n_clusters	the number of clusters that should be generated
weight_range	the range in which the weights for the structural equation model (SEM) can lie
lin_distribution_type	the linear distributions offered by gcastle [Zha+21b] can be gaussian, exponential or gumbel
nonlin_distribution_type	nonlinear distributions offered by gcastle [Zha+21b] can be multilayer perceptron (mlp), mutual information machine (mim) or gaussian process (gp)
sample_size	sample size for a configuration
seed	seed for the simulation
node_names	custom node names for the simulation graphs
noise_scale	the noise scale in the SEM models
alpha	the significance level for the CI test if $p > \alpha$, an edge gets deleted
sid	whether or not to compute the structural intervention distance from causal discovery toolbox, it is expensive to compute
indep_test	the CI test to be used, from causallearn package, can be 'fisherz' (Fisher-z) , 'chisq' (Chi-squared), 'gsq' (G-squared) or 'kci' (kernel CI test)
cluster_method	the method used to compute the clustering for the C-DAG, can be 'cdag' or 'dag', explained in Chapter A.2.1

Bibliography

[Wri18] Sewall Wright. "On the nature of size factors". In: *Genetics* 3.4 (1918), p. 367.

[Fis+21] Ronald Aylmer Fisher et al. "On the Probable Error of a Coefficient of Correlation Deduced from a Small Sample." In: (1921).

[Wri34] Sewall Wright. "The method of path coefficients". In: *The annals of mathematical statistics* 5.3 (1934), pp. 161–215.

[ER59] P ERDdS and A R&wi. "On random graphs I". In: *Publ. math. debrecen* 6.290-297 (1959), p. 18.

[Gil59] Edgar N Gilbert. "Random graphs". In: *The Annals of Mathematical Statistics* 30.4 (1959), pp. 1141–1144.

[Bar77] Jon Barwise. "An introduction to first-order logic". In: *Studies in Logic and the Foundations of Mathematics.* Vol. 90. Elsevier, 1977, pp. 5–46.

[Lau96] Steffen L Lauritzen. *Graphical models.* Vol. 17. Clarendon Press, 1996.

[AMP97] Steen A Andersson, David Madigan, and Michael D Perlman. "A characterization of Markov equivalence classes for acyclic digraphs". In: *The Annals of Statistics* 25.2 (1997), pp. 505–541.

[Ric98] Thomas S Richardson. "Chain graphs and symmetric associations". In: *Learning in graphical models.* Springer, 1998, pp. 231–259.

[Spi+00] Peter Spirtes et al. *Causation, prediction, and search.* MIT press, 2000.

[Chi02] David Maxwell Chickering. "Optimal structure identification with greedy search". In: *Journal of machine learning research* 3.Nov (2002), pp. 507–554.

[LR02] Steffen L Lauritzen and Thomas S Richardson. "Chain graph models and their causal interpretations". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 64.3 (2002), pp. 321–348.

[RS02] Thomas Richardson and Peter Spirtes. "Ancestral graph Markov models". In: *The Annals of Statistics* 30.4 (2002), pp. 962–1030.

[Rub05] Donald B Rubin. "Causal inference using potential outcomes: Design, modeling, decisions". In: *Journal of the American Statistical Association* 100.469 (2005), pp. 322–331.

[SP06] Ilya Shpitser and Judea Pearl. “Identification of joint interventional distributions in recursive semi-Markovian causal models”. In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006, p. 1219.

[KB07] Markus Kalisch and Peter Bühlman. “Estimating high-dimensional directed acyclic graphs with the PC-algorithm.” In: *Journal of Machine Learning Research* 8.3 (2007).

[Ack+08] B Ackley et al. “Evidence based nursing care guidelines”. In: *Medical Surgical Interventions. Mosby Elsevier, syf* 15 (2008).

[Zha08a] Jiji Zhang. “Causal reasoning with ancestral graphs”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1437–1474.

[Zha08b] Jiji Zhang. “On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias”. In: *Artificial Intelligence* 172.16-17 (2008), pp. 1873–1896.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[MKB09] Marloes H Maathuis, Markus Kalisch, and Peter Bühlmann. “Estimating high-dimensional intervention effects from observational data”. In: *The Annals of Statistics Vol.37, No. 6A* (2009), pp. 3133–3164. DOI: `10.1214/09-AOS685`.

[Pea09] Judea Pearl. *Causality*. Cambridge university press, 2009.

[Col+12a] Diego Colombo et al. “Learning high-dimensional directed acyclic graphs with latent and selection variables”. In: *The Annals of Statistics* (2012), pp. 294–321.

[Col+12b] Diego Colombo et al. “Supplement to ’Learning high-dimensional directed acyclic graphs with latent and selection variables’”. In: (2012). DOI: `10.1214/11-AOS940SUPP`. URL: `https://dx.doi.org/10.1214/11-AOS940SUPP`.

[HB12] Alain Hauser and Peter Bühlmann. “Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 2409–2464.

[Zha+12] Kun Zhang et al. “Kernel-based conditional independence test and application in causal discovery”. In: *arXiv preprint arXiv:1202.3775* (2012).

[HD13] Naftali Harris and Mathias Drton. “PC algorithm for nonparanormal graphical models.” In: *Journal of Machine Learning Research* 14.11 (2013).

[Mee13] Christopher Meek. “Causal inference and causal explanation with background knowledge”. In: *arXiv preprint arXiv:1302.4972* (2013).

[RR13] Thomas S Richardson and James M Robins. “Single world intervention graphs (SWIGs): A unification of the counterfactual and graphical approaches to causality”. In: *Center for the Statistics and the Social Sciences, University of Washington Series. Working Paper* 128.30 (2013), p. 2013.

[SMR13] Peter L. Spirtes, Christopher Meek, and Thomas S. Richardson. *Causal Inference in the Presence of Latent Variables and Selection Bias*. 2013. arXiv: `1302.4983 [cs.AI]`.

[CM+14] Diego Colombo, Marloes H Maathuis, et al. “Order-independent constraint-based causal structure learning.” In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 3741–3782.

[Shi14] Shohei Shimizu. “LiNGAM: Non-Gaussian methods for estimating causal structures”. In: *Behaviormetrika* 41 (2014), pp. 65–98.

[HTW15] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[PB15] Jonas Peters and Peter Bühlmann. “Structural intervention distance for evaluating causal graphs”. In: *Neural computation* 27.3 (2015), pp. 771–799.

[Rot+15] Dominik Rothenhäusler et al. “BACKSHIFT: Learning causal cyclic graphs from unknown shift interventions”. In: *Advances in Neural Information Processing Systems* 28 (2015).

[BP16] Elias Bareinboim and Judea Pearl. “Causal inference and the data-fusion problem”. In: *Proceedings of the National Academy of Sciences* 113.27 (2016), pp. 7345–7352.

[Le+16] Thuc Duy Le et al. “A fast PC algorithm for high dimensional causal discovery with multi-core PCs”. In: *IEEE/ACM transactions on computational biology and bioinformatics* 16.5 (2016), pp. 1483–1495.

[NMR17] Preetam Nandy, Marloes H Maathuis, and Thomas S Richardson. “Estimating the effect of joint interventions from observational data in sparse high-dimensional settings”. In: *The Annals of Statistics 45 (2)* (2017), pp. 647–674. DOI: `10.1214/16-AOS1462`.

[PJS17] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[HMM18] Christina Heinze-Deml, Marloes H Maathuis, and Nicolai Meinshausen. “Causal structure learning”. In: *Annual Review of Statistics and Its Application* 5 (2018), pp. 371–391.

[Zhe+18] Xun Zheng et al. “Dags with no tears: Continuous optimization for structure learning”. In: *Advances in neural information processing systems* 31 (2018).

[GZS19] Clark Glymour, Kun Zhang, and Peter Spirtes. “Review of causal discovery methods based on graphical models”. In: *Frontiers in genetics* 10 (2019), p. 524.

[ASC20] Bryan Andrews, Peter Spirtes, and Gregory F Cooper. “On the completeness of causal discovery in the presence of latent confounding with tiered background knowledge”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 4002–4011.

[HE20] Zhongyi Hu and Robin Evans. “Faster algorithms for Markov equivalence”. In: *Conference on Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 739–748.

[RSW21] Alexander Reisach, Christof Seiler, and Sebastian Weichwald. “Beware of the simulated dag! causal discovery benchmarks may be easy to game”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 27772–27784.

[Zha+21a] Kai Zhang et al. “A Fast PC Algorithm with Reversed-order Pruning and A Parallelization Strategy”. In: *arXiv preprint arXiv:2109.04626* (2021).

[Zha+21b] Keli Zhang et al. *gCastle: A Python Toolbox for Causal Discovery*. 2021. arXiv: `2111.15155 [cs.LG]`.

[Adi+22] Riddhiman Adib et al. “CKH: Causal Knowledge Hierarchy for Estimating Structural Causal Models from Data and Priors”. In: *arXiv preprint arXiv:2204.13775* (2022).

[ADG22] Charles K Assaad, Emilie Devijver, and Eric Gaussier. “Survey and evaluation of causal discovery methods for time series”. In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 767–819.

[Bro+22] Philippe Brouillard et al. “Typing assumptions improve identification in causal discovery”. In: *Conference on Causal Learning and Reasoning*. PMLR. 2022, pp. 162–177.

[Fan+22] Zhuangyan Fang et al. *On the Representation of Causal Background Knowledge and its Applications in Causal Inference*. 2022. arXiv: `2207.05067 [cs.AI]`.

[Kil22] Niki Kilbertus. “Causality lecture notes SS22 at Technical University Munich”. In: (2022).

[KLC22] Steven Kleinegesse, Andrew R Lawrence, and Hana Chockler. “Domain Knowledge in A*-Based Causal Discovery”. In: *arXiv preprint arXiv:2208.08247* (2022).

[WQZ22] Tian-Zuo Wang, Tian Qin, and Zhi-Hua Zhou. “Sound and complete causal identification with latent variables given local background knowledge”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 10325–10338.

[ZOS22] Alessio Zanga, Elif Ozkirimli, and Fabio Stella. “A survey on causal discovery: theory and practice”. In: *International Journal of Approximate Reasoning* 151 (2022), pp. 101–129.

[Ana+23] Tara V. Anand et al. “Causal Effect Identification in Cluster DAGs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.10 (June 2023), pp. 12172–12179. DOI: `10.1609/aaai.v37i10.26435`. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/26435`.

[BD23] Christine W Bang and Vanessa Didelez. “Do we become wiser with time? On causal equivalence with tiered background knowledge”. In: *arXiv preprint arXiv:2306.01638* (2023).

[CRT23] Jawad Chowdhury, Rezaur Rashid, and Gabriel Terejanu. “Evaluation of Induced Expert Knowledge in Causal Structure Learning by NOTEARS”. In: *arXiv preprint arXiv:2301.01817* (2023).

[CGK23] Anthony C Constantinou, Zhigao Guo, and Neville K Kitson. “The impact of prior knowledge on causal structure learning”. In: *Knowledge and Information Systems* (2023), pp. 1–50.

[GCL23] Shantanu Gupta, David Childers, and Zachary C Lipton. “Local Causal Discovery for Estimating Causal Effects”. In: *arXiv preprint arXiv:2302.08070* (2023).

[HG23] Uzma Hasan and Md Osman Gani. “KGS: Causal Discovery Using Knowledge-guided Greedy Equivalence Search”. In: *arXiv preprint arXiv:2304.05493* (2023).

[HHG23] Uzma Hasan, Emam Hossain, and Md Osman Gani. “A Survey on Causal Discovery Methods for Temporal and Non-Temporal Data”. In: *arXiv preprint arXiv:2303.15027* (2023).

Index